**Paulo Nei Cruz Filho**

# *Teste de Software Baseado em Perturbação de Dados Dirigida por Padrões*

Curitiba
2007

**Paulo Nei Cruz Filho**

# *Teste de Software Baseado em Perturbação de Dados Dirigida por Padrões*

Dissertação de mestrado. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná

Orientadora: Prof.ª Dr.ª Silvia Regina Vergilio

UNIVERSIDADE FEDERAL DO PARANÁ

Curitiba
2007

# *Resumo*

A eXtensible Markup Language (XML) tem sido adotada como o principal meio para a comunicação de componentes que trocam mensagens em uma rede. O projeto de vocabulários XML que definem como os componentes se comunicam é um fator determinante para a construção dos programas; entretanto as formas de definição desses vocabulários (pelo uso de definições DTD ou documentos XML *Schema*, por exemplo) apresentam muitas limitações. Surgiram então propostas do uso da Unified Modeling Language (UML) para o projeto dos vocabulários XML, com o objetivo de otimizar essa tarefa. Outra proposta para auxiliar o projeto de programas que utilizam XML foi o desenvolvimento de padrões de software para XML. Entretanto os padrões costumam não se beneficiar do uso da UML aplicada aos modelos XML. Esse trabalho explora a utilização de padrões XML baseados em UML, fazendo uso de técnicas de mapeamento UML/XML. São analisados dois grupos de padrões: os padrões de projeto de documentos e os padrões estruturais. Os padrões de projeto buscam mapear padrões da comunidade de software orientado a objetos, analisando benefícios e limitações. Os padrões estruturais têm o propósito de analisar a semântica dos modelos XML. Além de facilitar a escrita de documentos XML e o entendimento das aplicações, os padrões podem ser também utilizados para propor uma nova abordagem de teste baseado em perturbação que é dirigida por padrões XML. O teste baseado em perturbação tem sido aplicado com sucesso no teste de *Web Services*. Operadores de perturbação modificam mensagens XML, considerando seus aspectos sintáticos; as mensagens perturbadas são então usadas como dado de teste. Entretanto o conjunto de operadores existentes não é completo. A nova abordagem dirigida a padrões permite considerar também aspectos semânticos e revelar outros tipos de defeitos. A abordagem de perturbação de dados dirigida por padrões aqui proposta, foi implementada em uma ferramenta e utilizada em um experimento. Os resultados mostram a validade e aplicabilidade da abordagem, bem como sua habilidade em revelar defeitos.

# *Abstract*

eXtensible Markup Language (XML) has been adopted as the main means for communication of components that send messages in a network. The project of XML vocabularies that define how the components communicate is a decisive factor for the construction of programs; however the means of definition of those vocabularies (by using DTD definitions or XML Schema documents, for example) present some limitations. The use of Unified Modeling Language (UML) for the project of XML vocabularies has been proposed to ease this task. Another proposal to aid the project of programs that use XML was the development of software patterns for XML. However the current patterns generally do not benefit from the use of UML models. This work explores the use of UML based XML patterns, making use of UML/XML mapping techniques. Two groups of patterns are analyzed: the document design patterns and the structural ones. The design patterns map patterns of the object oriented software community, analyzing benefits and limitations. The structural patterns allow the analysis of the semantics of XML models, and help the creation of XML documents. Moreover, the patterns can be used to introduce a new testing approach based on patterns driven data perturbation. Perturbation based test has been successfully applied for *Web Services* testing. Perturbation operators had been proposed to modify XML messages, considering their syntactic aspects; the modified messages are then used as test data. However the set of existing operators is not complete. The new pattern driven approach considers semantic aspects and generates test data related to complementary types of faults. The pattern driven data perturbation approach was implemented in a tool and used in an experiment. The results show the validity and applicability of the approach, as well as its ability for revealing faults.

# *Conteúdo*

# *Lista de Figuras*

# *Lista de Tabelas*

# *Lista de Listagens*

*Computadores são para a computação o que instrumentos são para a música... Leonardo da Vinci chamou a música de escultura do invisível e sua frase é ainda mais apropriada como uma descrição do software.*
***A. Kay***

# 1 Introdução

Neste capítulo são apresentados o contexto no qual se insere esse trabalho, sua motivação e contribuições, os objetivos a serem alcançados e a organização dessa dissertação.

## 1.1 Contexto

Nos últimos anos a eXtensible Markup Language (XML) tornou-se o principal meio para a comunicação de dados entre aplicações na Internet. A XML define um formato de texto simples e muito flexível. Originalmente ela foi projetada para atender aos desafios da publicação eletrônica em larga escala. XML assumiu um importante papel na troca de uma grande variedade de dados na Web e em muitos outros lugares [W3C05]. Um fator central para a troca de dados é o projeto dos vocabulários XML (por meio de documentos XML *Schema*, por exemplo), os quais permitem que diversos componentes diferentes se comuniquem. Entretanto, as formas de definição desses vocabulários apresentam muitas limitações.

Os vocabulários definem os elementos que fazem parte da comunicação, assim como seus relacionamentos. Um aspecto muito importante a ser considerado é que os vocabulários XML têm o poder de expressar a semântica envolvida na comunicação [Car01]. A semântica determina um modelo de como os termos podem se relacionar entre si. A semântica também pode especificar uma estrutura de classificação entre os conceitos, assim como especialização dos termos.

Entretanto a maioria dos vocabulários projetados atualmente não faz um bom uso do poder de expressão semântico que a XML permite. Alguns vocabulários se restringem a somente uma lista de termos. Felizmente existem pesquisas relacionadas com esse assunto, que tentam apresentar metodologias que melhorem a construção dos vocabulários. Para auxiliar o projeto de vocabulários XML, foram desenvolvidas técnicas de modelagem com o uso de modelos UML, podendo-se citar [BCFK99, BGR02, Car01].

Os modelos UML têm o poder de capturar a semântica envolvida no domínio modelado. A modelagem de vocabulários XML com o uso de modelos UML traz muitas vantagens. A UML apresenta recursos que ajudam o projetista a criar modelos que tendem a ser mais expressivos em termos sintáticos e semânticos. Por outro lado, o projeto de vocabulários XML utilizando-se o formato textual (com uma DTD, por exemplo) é muito mais árduo, difícil de ser visualizado e tende a desprezar a expressão da semântica dos modelos. As técnicas de modelagem permitem realizar um mapeamento entre modelos XML e UML.

Outra proposta para auxiliar o projeto de aplicações XML foi o desenvolvimento de padrões de software específicos para XML. Os padrões podem ter diversas naturezas, de acordo com o domínio de sua aplicação. Existem padrões que são voltados para o projeto de documentos XML, tendo por objetivo auxiliar a estruturação da informação para o projeto de documentos DTD e XML *Schemas* [Lai05]. Esses padrões são relacionados especificamente com os recursos da linguagem XML, de forma independente do projeto da implementação do programa que usa XML. Outro grupo de padrões que se destaca são os padrões para o projeto de aplicativos que usam XML [Pon01]. Nesse contexto, os padrões descrevem problemas e soluções recorrentes no projeto de aplicações XML, sendo bastante relacionados aos padrões de projeto do software orientado a objetos [Lar98].

Entretanto existem muitas limitações na definição dos padrões XML, pois em geral são utilizados somente recursos textuais da linguagem XML, resultando em padrões dependentes da linguagem e centrados em particularidades sintáticas. Nesse trabalho é introduzida a proposta do uso de modelos UML para a definição de padrões XML específicos. Para que isso seja possível, são utilizadas as técnicas de mapeamento de UML para vocabulários XML. Os primeiros padrões de software surgiram na comunidade de software orientado a objetos, onde os modelos UML foram usados com sucesso para expressar suas estruturas [Lar98, GHJV95]. Assim, espera-se que o uso dos modelos UML para expressar estruturas de documentos XML possa ajudar o projeto de padrões para XML.

A Engenharia de Software busca a produção de software com qualidade. Para que se possa construir software de qualidade são necessárias atividades de verificação e validação, onde a atividade de teste contribui fortemente. Com o aumento do uso de tecnologias baseadas em XML, o desenvolvimento de técnicas de teste específicas para aplicações que utilizam essas tecnologias tem ganhado importância em pesquisas recentes [Alm05, AS06, OL01, OW04, OWL05].

*Web Services* são uma tecnologia que permite interoperabilidade na comunicação ponto-a-ponto sobre uma rede. A comunicação é feita por meio de troca de mensagens XML encapsuladas em um protocolo específico. Essa característica introduz complexidade ao teste da comunicação

sob XML, dado que técnicas tradicionais de teste não podem ser aplicadas diretamente. A técnica de perturbação de dados foi inicialmente explorada por Offutt e Wuzhi [OW04], no contexto de teste de aplicações que trocam mensagens XML. O princípio da técnica consiste em modificar uma mensagem XML que é utilizada na comunicação de componentes, segundo alguma regra, a fim de enviar a mensagem modificada, para se analisar a resposta. A mudança na mensagem é realizada por meio de operadores de perturbação, que têm por objetivo revelar defeitos específicos. A perturbação de dados pode auxiliar o teste de *Web Services*, tecnologia que vem sendo cada vez mais utilizada nos últimos anos [Blo02].

Apesar das técnicas de testes desenvolvidas propiciarem bons resultados quando aplicadas ao teste de componentes que usam vocabulários XML, os operadores de perturbação definidos por Offutt e Wuzhi analisam somente informações sintáticas do modelo. Dessa maneira a capacidade de expressão semântica que os modelos XML podem prover não é explorada. Nesse trabalho será apresentada uma proposta que contribui para diminuir essa limitação.

## 1.2 Motivação

Dado o contexto descrito, podem-se citar as principais motivações que guiam esse trabalho:

- A importância que a XML atingiu para a comunicação de dados.
- A necessidade de promover o projeto de vocabulários XML com maior qualidade, por meio de técnicas que modelem a semântica envolvida na comunicação.
- A pesquisa do potencial apresentado pelo uso da UML no auxílio ao desenvolvimento de aplicativos XML, com suas diversas contribuições.
- O estudo de padrões de software que possam ser aplicados ao desenvolvimento de software que utiliza XML, à definição dos vocabulários XML, ao projeto dos aplicativos que utilizam XML, ao teste de programas que usam XML, etc.
- A necessidade de desenvolver técnicas de teste de aplicações que se comunicam por meio do uso de mensagens XML, como *Web Services*, os quais paresentam uma grande utilização e ao mesmo tempo desafios para seu teste efetivo.
- A necessidade de explorar o poder de expressão dos vocabulários XML para melhorar a produção de dados de teste
- O fato de a técnica de perturbação de dados possuir limitações quanto à análise semântica, para as quais o uso de padrões XML pode contribuir. Os operadores de

perturbação utilizados nessa técnica não consideram aspectos semânticos do vocabulário XML.

- A necessidade de uma ferramenta para a aplicação prática da técnica de perturbação de dados ao teste de *Web Services* e para a avaliação dos conceitos teóricos desenvolvidos.

## 1.3 Objetivos

Essa pesquisa levou em consideração os seguintes objetivos:

- Analisar a maneira como a UML pode ajudar na definição de vocabulários XML, considerando as diversas propostas encontradas na literatura; e assim escolher o modo de mapeamento entre os domínios UML/XML que mais se alinha à motivação.
- Avaliar padrões de software aplicados ao contexto da modelagem XML, verificando como os padrões podem ajudar nas diversas aplicações da linguagem XML
- Explorar uma técnica de perturbação de dados que possa utilizar padrões XML que utilizam uma técnica de mapeamento UML/XML, em busca da geração de dados de teste que considerem aspectos semânticos do vocabulário XML envolvido na comunicação de componentes.
- Desenvolver uma ferramenta que possa aplicar os aspectos teóricos da perturbação de dados e da técnica proposta ao contexto da pesquisa, no teste de *Web Services*. Utilizar a ferramenta em um ambiente real, possibilitando avaliar as vantagens da técnica proposta, assim como suas limitações.

## 1.4 Organização do Trabalho

O restante dessa dissertação está organizado da seguinte forma: os Capítulos 2, 3 e 4 apresentam uma revisão bibliográfica, onde são vistos os conceitos que possibilitam compreender o contexto da proposta. O Capítulo 2 trata do uso da UML para modelar vocabulários XML, apresentando o estado da arte das abordagens disponíveis; o Capítulo 3 apresenta os trabalhos existentes sobre padrões em XML; o Capítulo 4 introduz conceitos de teste de software e técnicas relacionadas com as propostas de melhorias vistas nos capítulos seguintes.

Os Capítulos 5 e 6 apresentam a proposta da definição de padrões XML com o uso de modelos UML. No Capítulo 5 é visto o conceito de padrões de projeto; no Capítulo 6 são apresentados os padrões estruturais propostos.

O Capítulo 7 apresenta uma aplicação dos padrões propostos no teste de software. É introduzido o conceito de perturbação de dados dirigida por padrões. Em seguida, no Capítulo 8, é descrita uma ferramenta para testes baseados nesse tipo de perturbação. Um estudo de caso realizado com a ferramenta é mostrado no Capítulo 9.

No Capítulo 10 são apresentadas as conclusões e trabalhos futuros. O trabalho contém três apêndices: Apêndice A, onde são apresentados algoritmos utilizados na perturbação de dados; Apêndice B, no qual são mostrados exemplos de esquemas XML; e Apêndice C, onde são mostradas telas da ferramenta para testes desenvolvida.

*Vendo o que é antigo é que se aprende o que é novo.*
***Provérbio chinês***

# *2 O Uso da UML para Modelar Vocabulários XML*

O desenvolvimento de software é um campo que ainda necessita de muita pesquisa. As notações e vocabulários da área ainda estão sendo desenvolvidos, encontrando-se em evolução nos últimos anos. Têm-se várias maneiras de descrever os programas, os projetos de bancos de dados e os modelos para desenvolvimento de software.

A questão chave desses vocabulários é a capacidade de expressar visualmente conceitos que não são visuais. Na história da humanidade, os primeiros exemplos do uso dessa idéia encontram-se no desenvolvimento de ideogramas e alfabetos, capazes de reter o conhecimento em um modelo e transmiti-lo às outras pessoas. Na área de desenvolvimento de software, as linguagens definidas para tal fim são muitas, passando pelos vocabulários que expressam a execução de programas (como C ou Java) até os vocabulários que são usados para projetar os sistemas, como a Unified Modeling Language (UML) [BRJ99]. Dessa maneira, tem-se uma "Torre de Babel" de linguagens para o desenvolvimento de software.

Para que possa haver comunicação efetiva em meio a tantas formas de expressar o desenvolvimento de software, é necessário "traduzir" os conceitos de um vocabulário para outro. Uma importante aplicação dessa "tradução" entre modelos diferentes se dá no relacionamento de modelos UML e modelos XML. A modelagem de vocabulários para aplicações que usam XML pode obter muitas vantagens com o mapeamento para UML.

Esse capítulo trata dessas transformações entre os domínios XML e UML, analisando o estado da arte das técnicas disponíveis. Em especial é considerada a metodologia de tradução adotada [Car01]. Inicialmente são discutidos aspectos fundamentais sobre a linguagem XML e sobre a definição de XML *Schemas*; os conceitos apresentados permitem a visão do contexto ao qual é aplicado o mapeamento XML para UML.

## 2.1 Aspectos importantes da eXtensible Markup Language

A XML descende da SGML, a Standard *Generalized Markup Language*, adotada como um padrão ISO (8879) em 1986. A SGML é muito poderosa e obteve sucesso em uso para aplicações militares, governamentais e aeroespaciais (setores onde é preciso gerenciar documentos técnicos com centenas de páginas). Pode-se citar a HTML como o maior sucesso de aplicação da SGML. Mas a HTML constitui-se de um subconjunto bastante limitado, com *tags* pré-definidas. Além disso, a HTML mostrou-se muito limitada para auxiliar a troca de dados entre aplicações na *Web* [HM04].

Surgiu então a necessidade de uma linguagem para auxiliar a troca de dados na Internet, que possibilitasse a comunicação entre plataformas diferentes. Devido às suas vantagens, a SGML foi a linguagem escolhida como base para solucionar esse problema. Entretanto a SGML é extremamente complicada, de forma que nenhum software implementou completamente sua especificação, que possui mais de 150 páginas bastantes técnicas [ISO86]. O que se implementou foram subconjuntos da SGML. Dessa maneira foi desenvolvida a XML, com o propósito de se obter a maior parte do poder da SGML e reduzir a complexidade. O resultado foi a XML 1.0, lançada em fevereiro de 1998 [W3C04.b]. Imediatamente a XML obteve um grande sucesso, sendo adotada por desenvolvedores das mais diversas áreas.

XML define uma sintaxe para marcação de dados com *tags* humanamente legíveis. O formato é flexível o bastante para poder ser usado em diversos domínios de aplicação, como comércio eletrônico, música, biologia e muitos outros [HM04].

Existe uma grande biblioteca de software para manipular documentos XML, com um grande número de opções em software livre. Com o uso das bibliotecas, o programador pode abstrair muitos detalhes da linguagem, centrando-se na aplicação que usa XML.

XML é uma linguagem de meta-marcação para documentos de texto. Os dados são armazenados como *strings* de texto. A base da marcação é representada por elementos. Aos elementos, podem estar associados atributos. Superficialmente, um documento XML se parece com um documento HTML. Entretanto, como XML é uma linguagem de meta-marcação, não possui um conjunto fixo de *tags*. Isso permite que os desenvolvedores definam os elementos que precisam.

Existe uma gramática para XML, que diz como as *tags* devem ser organizadas, quais nomes são permitidos, como os atributos se relacionam aos elementos, entre outras características. Um documento que satisfaça à gramática é dito bem formado.

Uma aplicação XML é definida como o conjunto de *tags* especificadas para uma aplicação (e não um software de aplicação que usa XML) [HM04]. Esse termo pode ser confuso, devido à ambigüidade da palavra aplicação.

A marcação de um documento XML mostra como os elementos estão associados. Dessa maneira, existe uma semântica associada a uma estrutura de marcação. A marcação permitida para uma aplicação específica pode ser definida por um esquema. Um documento que satisfaz um determinado esquema é dito válido.

Existem muitas linguagens de definição de esquemas. A linguagem que teve maior aceitação inicial foi a *Document Type Definition* (DTD) [HM04]. Entretanto a sintaxe da DTD é bastante limitada, não permitindo a definição de tipos de dados, por exemplo. O XML *Schema* do W3C está avançando como uma alternativa mais robusta, permitindo uma sintaxe mais rica [W3C04.c].

### 2.1.1 W3C XML Schema

As DTDs mostraram-se bastante limitadas para a definição de muitas aplicações, que precisam de métodos mais expressivos de validação. O W3C desenvolveu o XML *Schema* com o objetivo de suprir essas necessidades. Os documentos XML *Schema* podem definir restrições bastante complexas sobre os elementos XML [W3C04.c].

O XML *Schema* provê um controle mais rico sobre o formato e sobre o tipo dos elementos e atributos, em comparação com a capacidade limitada dos DTDs. Assim que a XML começou a ser usada em aplicações centradas nos dados, um controle mais preciso sobre o conteúdo do texto dos elementos e atributos fez-se necessário.

Podem-se citar as seguintes características como avanços para o XML *Schema*: 1) definição de tipos de dados simples e complexos; 2) derivação de tipos e herança; 3) restrições sobre a ocorrência dos elementos; 4) controle do *namespace* dos atributos e elementos. Ao contrário das definições DTD, os documentos XML *Schemas* podem forçar regras específicas sobre o conteúdo dos elementos e atributos.

Dentre essas características, são de muita importância a herança e o controle sobre a ocorrência dos elementos. Veremos que durante o mapeamento dos modelos XML para modelos UML a capacidade de herança será muito explorada. O controle da ocorrência dos elementos

possibilitará um melhor mapeamento da multiplicidade das associações UML, que é mais limitado quando se usam DTDs.

Um grande número de tipos de dados foram definidos por padrão, podendo-se citar *string*, *integer*, *decimal* e *dateTime*. Além disso, a linguagem provê maneiras de definir novos tipos de dados, que são derivados dos tipos primitivos. Outra habilidade que o XML *Schema* possui é criar restrições mais explícitas sobre o número de elementos que podem aparecer como filhos de outros elementos. Também é possível definir a seqüência desses elementos.

Como os documentos XML estão sendo amplamente usados para a troca de dados entre as mais diferentes áreas, o uso de *namespaces* possibilitado pelo XML *Schema* é crucial. Os XML *Schemas* podem realizar validação sobre uma combinação de *namespaces*.

## 2.2 Mapeamento das linguagens XML e UML

A maioria dos vocabulários XML desenvolvidos define apenas uma lista de termos, sem uma semântica completa. Assim é comum encontrarmos apenas uma lista de termos de um vocabulário XML, sem saber como eles se relacionam. Esses vocabulários apresentam pouco significado no contexto da análise da comunicação entre aplicações XML.

Um problema dos vocabulários XML desenvolvidos dessa maneira é que o significado dos termos deve ser inserido no código da aplicação que processa o documento. Isso resulta em projetos inflexíveis. Outra abordagem para se definir a semântica do vocabulário é associá-los a descrições textuais, que são úteis somente para leitores humanos.

Levando em consideração esses aspectos, o uso da UML para definir vocabulários XML surge como uma alternativa que serve tanto para a representação do vocabulário para humanos, quanto para o processamento por aplicações computacionais. Uma grande vantagem que o uso da UML traz é a capacidade de captar a semântica do modelo XML. Essa capacidade será explorada nesse trabalho, com a proposta de definição de padrões estruturais. Esses padrões buscam revelar a semântica dos modelos, o que pode contribuir na análise automática dos documentos.

Existem muitas formas diferentes de se realizar essa integração entre os modelos UML e XML. A primeira tentativa de se realizar esse mapeamento foi feita por [BCFK99], num *White Paper* da Rational, onde são introduzidas extensões UML para a modelagem de elementos, atributos e grupos, entre outros.

Carlson [Car01] definiu uma metodologia fundamentada nas regras de transformação de UML para XML *Schema*, baseadas em XMI (*XML Metadata Interchange*[1]). A partir desse trabalho, foi desenvolvida uma ferramenta que realiza a transformação entre modelos UML e XML *Schemas*, disponível gratuitamente no *site* associado ao livro[2].

Bird et al [BGR02] propuseram um método baseado no projeto em três camadas de banco de dados. Nesse trabalho, o nível conceitual é representado usando notação UML padrão, anotada com restrições conceituais. O nível lógico é representado em UML usando-se um conjunto de estereótipos. O XML *Schema* representa o nível físico.

Podem-se citar ainda os trabalhos de Bernauer et al [BKK03], Freytag et al [FCS00] e Provost [Pro02.a], que apresentam propostas baseadas nos trabalhos anteriores. Essas propostas têm uma base semelhante à dos outros trabalhos, acrescida de revisões e propostas de melhorias. Bernauer et al apresentam uma comparação entre todos os métodos citados.

## 2.3 Mapeamento Proposto por Carlson

A seguir será mostrado como se realiza o mapeamento de modelos UML para modelos XML. O texto não pretende ser uma referência completa sobre o assunto, que de fato deu origem a um livro [Car01]. São tratados os pontos fundamentais no contexto desse trabalho. Dessa maneira, aspectos que não foram aplicados, como o mapeamento para DTD e perfis de extensão da UML não são abordados.

Outra observação deve ser feita em relação à ferramenta utilizada. Os conceitos de mapeamento UML/XML foram implementados na ferramenta hyperModel. Entretanto a implementação apresenta algumas mudanças em relação à abordagem do livro, pois alguns princípios da teoria precisaram ser reestruturados para permitir a implementação (em alguns aspectos, a teoria do livro foi bastante experimental; a ferramenta mostrou que alguns conceitos precisaram ser revisados).

---

[1] O XML Metadata Interchange é um padrão da OMG para a troca de informações sobre meta-dados por meio do uso da XML. O uso mais comum de XMI é na troca de modelos UML entre aplicações diferentes, embora seja possível usar XMI para a serialização de modelos de outras linguagens (meta-modelos) [OMG05].

[2] http://www.XMLModeling.com

Dessa maneira, sempre que houver diferenças entre a implementação da ferramenta e a estrutura teórica, será adotada a metodologia que de fato foi implementada no hyperModel, pois o tratamento dos modelos foi feito com essa ferramenta. Além disso, a implementação resolveu alguns problemas, cuja solução não foi apresentada no livro, ou foi apresentada de forma incompleta.

### 2.3.1 Mapeamento de Instâncias UML

Inicialmente é apresentado como se realizar o mapeamento de instâncias de modelos UML para instâncias de documentos XML. O objetivo é mostrar exemplos que permitam a compreensão de como será realizada a mudança. Em seguida, no item 2.3.2, é tratado o mapeamento de XML *Schemas*, que serão o principal foco de aplicação do mapeamento, pois definem o vocabulário XML. O mapeamento das instâncias é apresentado primeiro para mostrar exemplos concretos, que ajudam a compreender melhor os conceitos.

#### 2.3.1.1 Classes UML para Elementos XML

Nas classes, devido à natureza estrutural da XML, os elementos comportamentais (métodos) são desconsiderados. Em UML, uma classe serve como um contêiner para atributos e regras de associação. Dessa maneira, as classes são mapeadas para elementos XML, pois eles servem como contêineres para atributos e elementos filhos. Uma instância de classe UML é simplesmente mapeada para um elemento, onde o nome da *tag* XML é o mesmo nome da classe (respeitando-se as restrições de nomes válidos para XML).

#### 2.3.1.2 Herança

As instâncias XML não possuem uma forma específica para representar o conceito de herança. Para fazer o mapeamento, os elementos da superclasse são reproduzidos nas subclasses. A herança irá, portanto, ser aplicada na reprodução de atributos e associações nas classes herdeiras.

A definição do vocabulário XML a partir de uma DTD não possui nenhum mecanismo de especificação de herança. Já a definição a partir de um XML *Schema* inseriu mecanismos para permitir a expressão de relacionamentos de herança. O mapeamento de XML *Schemas* será analisado mais adiante.

#### 2.3.1.3 Atributos UML para Elementos XML

Os atributos de uma instância UML podem ser mapeados para XML como elementos filhos do elemento que representa a classe. Para ilustrar os conceitos vistos até agora, vamos ver o mapeamento do modelo mostrado na Figura 2. 1.

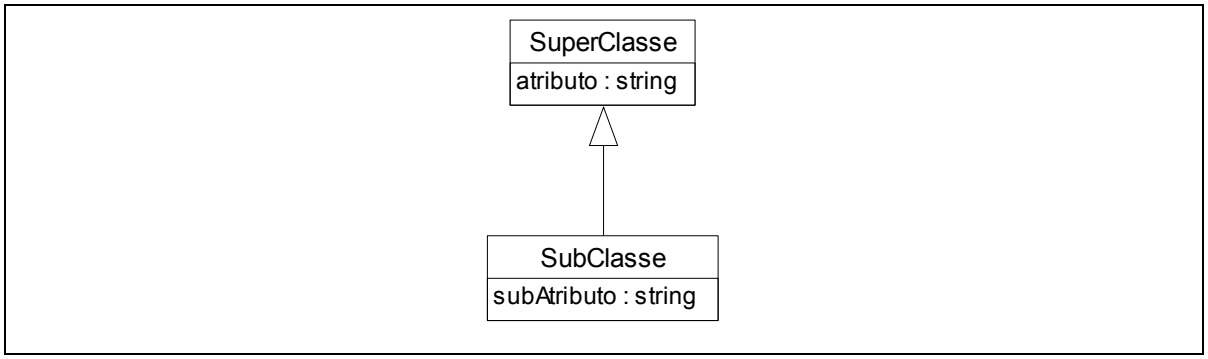



*Figura 2. 1 Modelo UML exemplo*

O mapeamento de uma instância da `SuperClasse`, com um `atributo`, é feito da seguinte forma:

```
<SuperClasse>
  <atributo>
    um atributo
  </atributo>
</SuperClasse>
```

O mapeamento de uma instância da `SubClasse`, com um `atributo` e um `subAtributo`, é mostrado a seguir. Observe como o atributo da superclasse foi herdado pela subclasse.

```
<SubClasse>
  <atributo>
    um atributo
  </atributo>

  <subAtributo>
    um sub-atributo
  </subAtributo>
</SubClasse>
```

#### 2.3.1.4 Atributos UML Para Atributos XML

Os atributos UML podem ser mapeados, também, para atributos XML. Para isso eles devem ser tipos de dados primitivos (como *string*). O mapeamento de uma instância da `SubClasse` da Figura 2.1, com o uso de atributos, pode ser feito da seguinte forma:

```
<SubClasse
  atributo="um_atributo" subAtributo="um_sub_atributo"
/>
```

Contudo essa forma de mapeamento apresenta algumas restrições:

1) o valor de um atributo XML é normalizado com relação a espaços em branco (tabulações, avanço de linha, retorno de carro, etc);

2) os atributos não são apropriados para valores de *strings* grandes (para leitores humanos)

3) um atributo UML que possua multiplicidade maior que um não pode ser mapeado para um atributo XML, pois não há um mecanismo que permita atributos com múltiplos valores (esse comportamento pode ser obtido com o uso de múltiplos elementos XML com o mesmo nome)

4) os atributos XML não são ordenados (os elementos XML podem ser ordenados, se for usado um XML *Schema* para definição do vocabulário)

A escolha do uso de elementos ou atributos em XML depende da natureza dos dados. Por exemplo, se o documento for visualizado por um ser humano, uma determinada ferramenta pode esconder os atributos das *tags* e tratá-los como metadados. Já para um processador automatizado, a distinção entre o uso de elementos ou atributos pode não fazer diferença.

O mapeamento de atributos UML para atributos XML pode ser obtido com a ferramenta hyperModel, apesar de não ser o comportamento padrão. Nesse trabalho, o mapeamento padrão será usar elementos XML para representar atributos da UML.

#### 2.3.1.5 Mapeamento de Associações e Composições UML

Apesar de Carlson ter mostrado que é possível mapear associações e composições de maneiras diferentes, sua ferramenta hyperModel não faz distinção entre o mapeamento de uma associação ou composição UML. Como esse trabalho trata dos aspectos estruturais dos correspondentes modelos UML, a forma como o mapeamento de associações ou composições é feito não é significativa para esse fim, podendo ser um fator a ser escolhido.

Nos futuros diagramas UML apresentados, o uso de associações ou composições será feito sem distinções. A notação de composição será usada somente para destacar o aspecto composicional dos relacionamentos. A expressão “associação” será usada como termo padrão nas descrições dos relacionamentos entre as classes.

A forma de mapeamento usada nesse trabalho segue o padrão da ferramenta hyperModel e será explicada a seguir. O nome da associação é mapeado como um elemento XML, de forma semelhante a que os atributos UML foram mapeados para elementos.

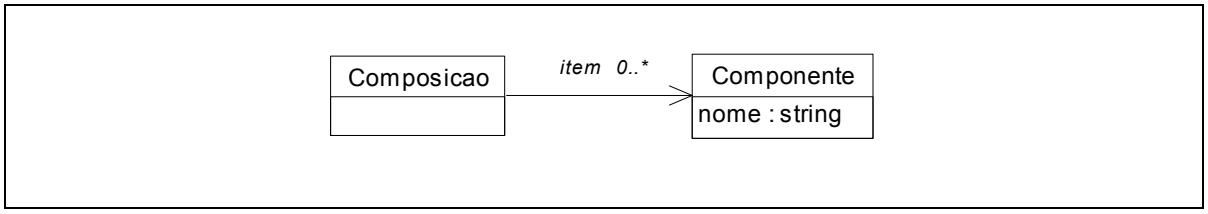



*Figura 2. 2 Exemplo de modelo UML para associação*

Considere o exemplo de diagrama UML apresentado na Figura 2. 2 . Uma instância que obedece a esse modelo é apresentada a seguir. Note o uso do elemento `item`, que representa a associação; observe também que existem dois itens, o que é permitido para a multiplicidade de zero ou mais vezes.

```
<Composicao>
  <item>
    <nome>
      um item
    </nome>
  </item>
  <item>
    <nome>
      outro item
    </nome>
  </item>
</Composicao>
```

Caso o nome da associação fosse omitido, poder-se-ia usar o nome da classe associada, como nesse exemplo:

```
<Composicao>
  <Componente>
    <nome>
      um componente
    </nome>
  </Componente>

  <Componente>
    <nome>
      outro componente
    </nome>
  </Componente>
</Composicao>
```

### 2.3.2 Mapeamento de XML *Schemas*

A seguir é mostrado como se realiza o mapeamento de um modelo UML para um vocabulário XML definido por um XML *Schema*.

#### 2.3.2.1 Classes

As classes UML são mapeadas para uma definição de um `ComplexType` no esquema. Além dessa definição, é declarado um elemento com o mesmo tipo do `ComplexType` definido. Todos os elementos são prefixados com o espaço de nomes “xs:”, que se refere ao *namespace* do XML *Schema*. O exemplo abaixo mostra a forma geral do mapeamento de uma classe:

```
<xs:element name="Classe" type="Classe"/>

<xs:complexType name="Classe">
  ...
</xs:complexType>
```

#### 2.3.2.2 Atributos

O `ComplexType` é definido usando-se o modelo de conteúdo `<sequence>`, que permite uma lista de elementos ordenados. Também se podem usar os modelos de conteúdo `<all>` e `<choice>`; entretanto será usado, como forma padrão, o modelo `<sequence>`, que permite maior controle no mapeamento (pois pode definir a ordem dos elementos) e também é o padrão da ferramenta hyperModel. Os atributos são mapeados como elementos filhos do elemento correspondente à classe. Uma classe que possua um `atributo` do tipo *string* é mapeada como:

```
<xs:element name="Classe" type="Classe"/>

<xs:complexType name="Classe">
  <xs:sequence>
    <xs:element name="atributo" type="xs:string" minOccurs="0"
     maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

Note que se pode definir a multiplicidade do `atributo`, com o uso de `minOccurs` e `maxOccurs`, que denotam o número mínimo e máximo de ocorrências daquele elemento.

#### 2.3.2.3 Associações

As associações são mapeadas de forma semelhante à forma como os atributos são mapeados para elementos. Nesse caso, o tipo do elemento é do mesmo tipo da classe à qual a associação faz referência. Considerando que se tenha a definição de `OutraClasse`, uma associação chamada `itens` de `Classe` para `OutraClasse` é mapeada como:

```
<xs:element name="Classe" type="Classe"/>
<xs:complexType name="Classe">
  <xs:sequence>
    <xs:element name="atributo" type="xs:string" minOccurs="0"
     maxOccurs="unbounded">
    </xs:element>
    <xs:element name="itens" type="OutraClasse" minOccurs="0"
     maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:element name="OutraClasse" type="OutraClasse"/>
<xs:complexType name="OutraClasse">
  ...
</xs:complexType>
```

Uma associação que não possua um nome é mapeada com o uso do atributo `ref`, em vez de `type`, sendo seu nome omitido:

```
<xs:complexType name="Classe">
  <xs:sequence>
    ...
    <xs:element ref="OutraClasse" minOccurs="0" maxOccurs="unbounded">
    </xs:element>
      ...
  </xs:sequence>
</xs:complexType>
```

#### 2.3.2.4 Herança

O mecanismo de herança do XML *Schema* é usado para mapear o relacionamento de herança UML. Uma classe do tipo `SubClasse`, que seja derivada do tipo `Classe` é definida da seguinte forma:

```
<xs:element name="SubClasse" type="SubClasse" substitutionGroup="Classe"/>

<xs:complexType name="SubClasse">
  <xs:complexContent>
    <xs:extension base="Classe">
      <xs:sequence>
        <xs:element name="subAtributo" type="xs:string">
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

A `SubClasse` é definida de forma semelhante a uma classe normal, com a diferença de que é indicada sua classe de extensão, pelo uso do atributo `base` do elemento `extension`. Isso permite que `SubClasse` herde todos os elementos definidos em `Classe`. Outra diferença é que a

declaração do elemento `SubClasse` usa o atributo `substitutionGroup`, indicando que toda vez que se puder usar um elemento do tipo `Classe`, também se poderá usar um elemento do tipo `SubClasse` em seu lugar.

### 2.3.3 Exemplo Completo

Para mostrar um exemplo da abordagem de Carlson a Figura 2. 3 apresenta uma comparação entre os domínios XML e UML. É apresentada uma classe que possui uma associação com uma hierarquia de herança, composta por uma superclasse e sua subclasse.

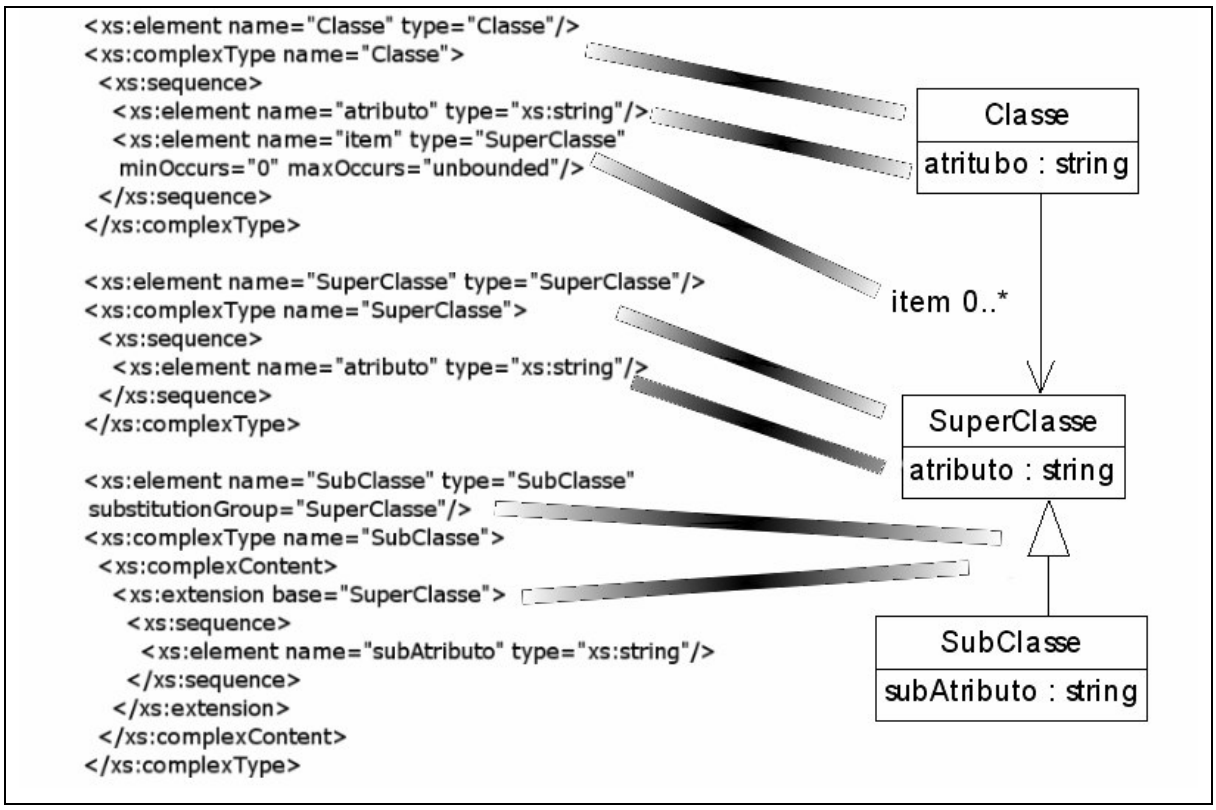



*Figura 2. 3 Exemplo para ilustrar o mapeamento completo do XML Schema*

## 2.3 Considerações Finais

Esse capítulo tratou de questões importantes sobre a XML, as quais são relevantes no contexto desse trabalho. Em especial, foram vistas técnicas de mapeamento entre as linguagens XML e UML, e mais detalhadamente o mapeamento proposto por Carlson, o qual será utilizado ao longo dessa proposta. Essa escolha foi realizada com base nos seguintes fatos:

1) a metodologia foi implementada (com pequenas mudanças) na ferramenta hyperModel, disponível para livre uso;

2) em seu livro [Car01], Carlson apresenta uma ampla documentação da metodologia, desenvolvida de forma bastante completa;

3) para fins de análise de padrões, outras propostas como a de Bird et al não apresentam uma estrutura que propicia uma análise de forma direta (dado que a proposta apresenta uma separação complexa dos modelos, com transformações entre camadas e o uso de restrições e um perfil UML específico para se obter o mapeamento final).

Outra construção que auxilia o desenvolvimento utilizando XML será vista no próximo capítulo: os padrões para XML. As técnicas de mapeamento XML/UML vistas nesse capítulo serão aplicadas na proposta de padrões XML baseados em UML, nos Capítulos 5 e 6.

*Cada padrão descreve um problema que ocorre de forma recorrente em um ambiente, e então descreve o centro da solução para aquele problema, de forma que você pode reutilizar essa solução um milhão de vezes, sem nunca repeti-la.*
***C. Alexander***

# *3 Padrões em XML*

As palavras presentes na citação acima revelam a motivação usada na primeira obra publicada sobre padrões, que foi escrita por Christopher Alexander, na área da Arquitetura [AIS77]. As idéias usadas na Arquitetura logo foram importadas para o mundo do desenvolvimento de software, o que abriu inúmeras possibilidades.

Esse capítulo apresenta trabalhos relacionados ao uso de padrões em XML. Os padrões mostrados são divididos em dois conjuntos: padrões voltados para o desenvolvimento de documentos XML e padrões para o projeto de aplicativos que usam XML. Para cada um desses grupos de padrões é apresentado um exemplo.

O reconhecimento de padrões é uma ferramenta fundamental em muitas áreas. Padrões surgem em inúmeros ambientes, podendo-se citar desde eventos da natureza, passando por fenômenos culturais, até a própria engenharia de software. Na engenharia de software, a prática de reconhecimento de padrões foi inicialmente utilizada para o projeto de software orientado a objetos. Esse capítulo mostra que os padrões podem ser utilizados com sucesso também em software baseado em XML.

## 3.1 Padrões de Software

O uso do termo “padrão” pode estar relacionado a diversos contextos. Um dos contextos mais conhecidos, na área de sistemas orientados a objetos, é proposto por Gamma *et al* [GHJV95]. Nesse âmbito, um padrão sistematicamente nomeia, motiva e explica um projeto geral para problemas recorrentes que surgem nos sistemas orientados a objetos. O padrão descreve o problema, a solução, conseqüências do uso do padrão, etc.

Como foi mencionado, o primeiro uso do conceito de padrões ocorreu na Arquitetura. Padrões foram usados para auxiliar o projeto de construções e cidades; esses padrões tratam de assuntos como melhores posições dos cômodos de uma casa, projetos de jardins e estrutura das estradas, entre outros.

No final da década de 80, os conceitos advindos da arquitetura começaram a ser pesquisados pela comunidade de software orientado a objetos. Os primeiros autores a publicarem um trabalho sobre esse assunto foram Ward Cunningham e Kent Beck [BC87]. Foram descritos cinco padrões referentes a projeto de aplicações em Smalltalk. Os benefícios do uso dos padrões foram rapidamente reconhecidos pela comunidade, que começou a escrever artigos, realizar seminários e publicar livros sobre o assunto. Um marco foi o lançamento do livro *Design Patterns* [GHJV95], notadamente a obra mais conhecida da área, que teve um papel muito importante na divulgação dos padrões, constituindo-se de uma referência constante até hoje.

## 3.2 Padrões para Documentos e Aplicativos

Os padrões podem ter diversas naturezas. A idéia mais comum refere-se aos padrões de projeto, que são os mais populares na comunidade de software orientado a objetos. Surgiram padrões de análise, padrões de processos e padrões organizacionais. Na área de XML, destacam-se padrões para o projeto de documentos XML e padrões para o projeto de aplicativos que usam XML. O termo “aplicativo” é usado aqui para referenciar o software que usa o documento XML; a palavra é usada para não causar ambigüidade com o termo “aplicação”, que no contexto XML designa um conjunto de *tags* específicas (cf. Capítulo 2).

Esse trabalho introduz a idéia do uso de modelos UML para detectar padrões estruturais XML. Também são analisados padrões de projeto de documentos XML, com o auxílio da UML. Alguns trabalhos usam os termos “padrões”, “padrões de projeto” e “padrões estruturais” (ou mistura entre esses termos) com significados diferentes [Arc00, Lai05, Pon01, Pro02.b]. Alguns termos usados e/ou definidos nesse trabalho podem apresentar significados diferentes do encontrado nas obras citadas (dado que não se encontra uma conformidade entre os autores). Para não causar ambigüidade, são utilizados os termos com os significados estabelecidos a seguir:

1) Padrões para documentos XML: preocupam-se apenas com o projeto dos documentos que utilizam a linguagem XML;

2) Padrões para aplicativos XML: são abstrações de padrões de projeto no nível do aplicativo que usa documentos XML;

Para compreender a diferença entre os padrões para documentos XML e os padrões para aplicativos XML, são apresentados a seguir descrições e exemplos de cada um desses tipos de padrões.

## 3.3 Padrões para Documentos XML

Os padrões de projeto de documentos XML têm por objetivo auxiliar a estruturação da informação para o projeto de DTD's e XML *Schemas*. Como exemplos de trabalhos que se enquadram nesse contexto, podem-se citar: [Lai05, Pro02.b]. Esses padrões são voltados especificamente para o uso dos recursos da linguagem XML.

São apresentadas soluções de propósito geral, independentes da implementação do aplicativo XML, sendo relacionadas somente aos documentos XML associados. Dessa maneira, os padrões de projeto de documentos são centrados na estrutura dos documentos XML.

A seguir é apresentado um exemplo de padrão de documento. Busca-se captar a idéia geral desse tipo de padrão, para comparar com outros tipos (a seguir) e com a proposta de padrões com base na UML.

### 3.3.1 Padrão Catch-All Element [Lai05]

O padrão baseia-se no uso de um elemento que trata de outros elementos desconhecidos no documento. O problema que o padrão resolve é o que ocorre quando usuários precisam ser capazes de inserir texto com marcação no documento XML, onde tal texto não é conhecido a priori. Um exemplo ocorre quando é necessário ter *tags* de marcação para apresentação dentro de um documento, como formatação de negrito, itálico, etc. Quando isso acontece, os processadores do documento XML podem não estar preparados para tratar essa situação.

O padrão permite o uso de elementos de outros tipos de documentos (com um *namespace* diferente), o que possibilita redução de custos e tempo, pois se pode reusar esquemas em vez de ter que desenvolvê-los, para ficarem de acordo com o processamento XML. Permitir que documentos usem outros esquemas garante um alto grau de flexibilidade.

A solução apresentada pelo padrão é criar um novo elemento, que servirá como um contêiner para os elementos do novo esquema. Por exemplo, um documento que descreve um carro pode permitir elementos que incluem elementos HTML:

```
<car>
  <model>Coupe</model>
  <year>1979</year>
  <comment>
    <xhtml:p xmlns:xhtml="http://www.w3.org/1999/xhtml">
    This car <xhtml:b>runs great!</xhtml:b>
  </comment>
</car>
```

Para que a inclusão do comentário com marcação seja possível, o XML *Schema* que define o vocabulário usa o elemento `any`, da seguinte forma:

```
<element name="car">
  <complexType>
    <sequence>
      <element name="model" type="string"/>
      <element name="year"   type="string"/>
      <element name="comment">
        <complexType>
          <sequence>
            <any
             namespace="http://www.w3.org/1999/xhtml"
             minOccurs="1" maxOccurs="unbounded"
             processContents="skip"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
```

Esse exemplo, bastante simples, mostra o uso de um *namespace* externo que é bastante comum. Entretanto o padrão permite que qualquer *namespace* diferente seja usado. Os elementos internos ao elemento “pega-tudo” (*catch-all*) podem pertencer a um *namespace* arbitrário. A *tag* do elemento “pega-tudo” funciona como uma advertência ao processador de XML, avisando que marcação advinda de outro *namespace* pode ser encontrada.

### 3.3.2 Uso da UML em padrões para Aplicativos

Uma proposta de definição de padrões para aplicativos, com o auxílio da UML foi apresentada por Provost [Pro02.b]. Nessa proposta são utilizados diagramas UML para abstrair documentos de esquemas XML em alto nível. Assim os elementos de um esquema são

representados em classes UML, as quais deixam mais explícitos os relacionamentos entre os elementos, pela visualização do diagrama de classes (em comparação aos relacionamentos mais obscuros que o uso apenas do esquema XML traria). A Figura 3.1 apresenta um exemplo de diagrama de classes UML para representar um esquema XML. Os detalhes dessa transformação não são explicados explicitamente pelo autor, que usa uma variação do mapeamento de Carlson [Car01].

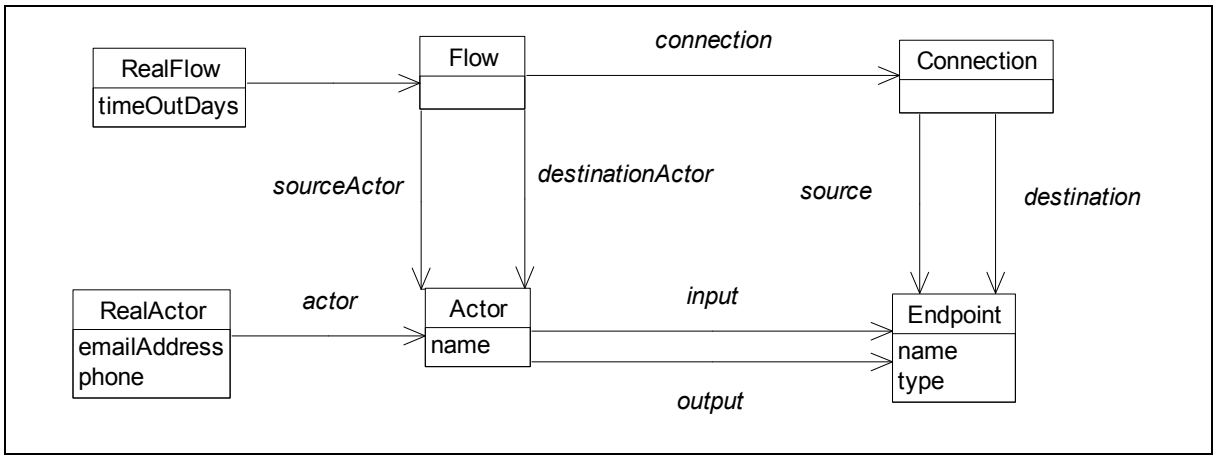



*Figura 3. 1: Diagrama de classes para representar esquema*

Os padrões apresenados por Provost são bastante semelhantes à exploração feita nessa pesquisa, que tentou mapear padrões de projeto conhecidos, utilizando a mudança de domínio UML/XML. A principal diferença é que Provost não faz um mapeamento de padrões existentes e também não define uma maneira formal para realizar a transformação. Nessa pesquisa é prestada atenção especial às dificuldades do mapeamento e à sua aplicabilidade, como será mostrado no Capítulo 5.

## 3.4 Padrões para Aplicativos XML

Esse segundo grupo de padrões constiu-se em maneiras de se abstrair o uso de XML no nível dos aplicativos. São descritos problemas e soluções recorrentes no projeto de aplicativos que processam documentos XML. Podem-se citar como exemplos de trabalhos que utilizam esse conceito: [Arc00, Pon01].

Por estarem direcionados ao projeto dos aplicativos, e não somente ao projeto do documento XML, esses padrões lembram muito os padrões de projeto tradicionais para software orientado a objetos. Nesse contexto, os padrões para aplicativos XML possuem natureza

comportamental, ao contrário dos padrões para documentos, que possuem somente a natureza estrutural. A seguir é apresentado um exemplo desse tipo de padrão.

### 3.4.1 Padrão In Out Tray [Pon01]

Esse padrão de projeto de aplicação XML organiza a atividade de componentes envolvidos no processo de obter e mostrar dados. Suponha que seja necessário obter dados de uma fonte, segundo algum critério definido dinamicamente (pelo usuário, por exemplo). Após obter os dados, é necessário apresentá-lo ao usuário. É desejável que o projeto seja flexível o suficiente para permitir que dois componentes sejam capazes de trocar dados sem formarem uma massa única de código.

O objetivo é desenvolver conexões entre os componentes, através das quais os dados possam trafegar, mantendo alta coesão e baixo acoplamento. Uma solução ingênua seria implementar um único componente para receber a requisição, procurar os dados, processar o que encontrou e gerar a saída para o usuário. Esse projeto não apresenta uma forma clara de troca de dados entre componentes internos. A solução força o desenvolvedor a entender o modelo de dados da fonte e também aumenta o acoplamento de componentes internos. Além disso, a flexibilidade e reusabilidade tornam-se baixas.

O padrão Int Out Tray apresenta uma forma de relacionar os componentes para permitir independência de linguagens e diferentes estruturas de dados XML coexistirem. São definidos três componentes computacionais e um arquivo XML. O primeiro componente (*In*) recebe dados contendo o critério de seleção dos dados. O segundo (*Worker*) recebe o critério de *In* e busca a informação no arquivo XML (o único requisito é: *Worker* precisa entender o formato da requisição e o formato do arquivo XML). *Worker* executa toda a lógica de negócio sobre a informação encontrada. Dessa maneira, tem-se uma separação entre o código de processamento das regras de negócio e o código de entrada e saída. Por fim, *Worker* envia o resultado do processamento para um terceiro componente, *Out*, que é responsável pela formatação para o usuário. Os relacionamentos entre os componentes são mostrados na Figura 3. 2.

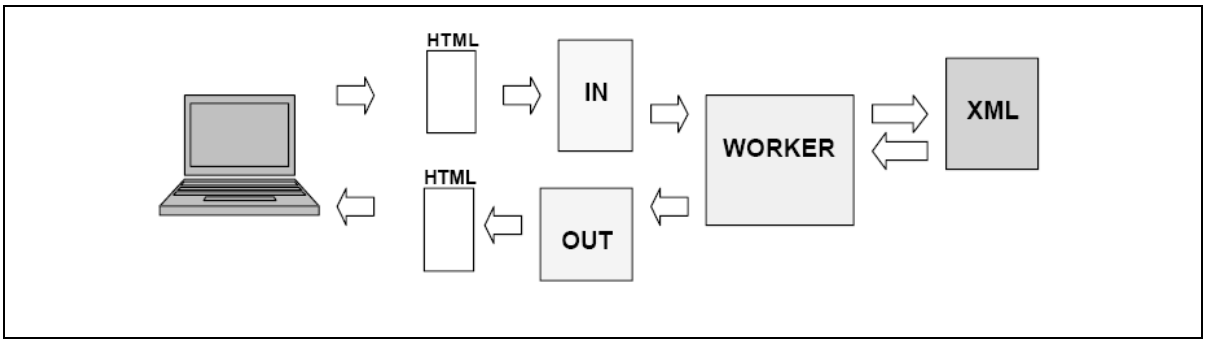



*Figura 3. 2: Componentes envolvidos no padrão In Out Tray*

Se for desejável receber dados de diferentes dispositivos de entrada e exibir o resultado em diferentes dispositivos de saída, é possível utilizar múltiplos componentes *In* e *Out*, um para cada dispositivo.

As conseqüências do uso desse padrão são:

1) Existe independência na representação dos dados;
2) O projeto final é flexível o suficiente para ser usado em diferentes circunstâncias;
3) A solução é modular, o que a torna fácil de usar, desenvolver e otimizar.

O padrão proposto lembra muito a arquitetura *Model-View-Controller* (MVC) [BMRS96]. Essa arquitetura teve grande aceitação no projeto de aplicações *Web*, principalmente com a popularização do uso do *framework* Struts, desenvolvido pela Apache Software Foundation [Apa05.a]. O paradigma MVC é uma forma de separar o projeto de uma aplicação em três partes: 1) o Modelo, responsável pelas regras de negócio que agem sobre os dados); 2) a Exibição, que trata da visualização da informação pelo usuário; 3) o Controlador, que interpreta os comandos (entrada) do usuário, transformando-os em requisições (conhecidas como ações) enviadas ao Modelo e/ou Exibição.

Apesar da simplicidade aparente do padrão, é muito comum encontrar aplicações onde não existe separação entre o código de apresentação e a lógica de negócios. A falta de organização dos módulos de software é característica principalmente em aplicações *Web*, nas quais existe uma grande tendência em se misturar a apresentação (*tags* HTML), com lógica de negócio (implementada em *servlets* ou *scripts*, por exemplo). Portanto, a aplicação de padrões de projeto que introduzem uma boa distinção entre esses módulos (componentes) é essencial em aplicações *Web* que usam XML.

## 3.5 Considerações Finais

Nesse capítulo foram mostrados trabalhos existentes sobre padrões em XML. Os padrões foram divididos em dois grupos: padrões para documentos e padrões para aplicativos. Um exemplo de cada um desses padrões foi mostrado. Os padrões para documentos são utilizados na estruturação dos esquemas que definem os vocabulários. Os padrões para aplicativos são aplicados no projeto dos sistemas que utilizam XML em alguma parte do processamento.

O Capítulo 5 apresenta padrões XML que fazem uso da UML (usando o mapeamento visto no capítulo anterior). A natureza desses padrões está relacionada aos padrões para documentos, fazendo uso de abstrações UML para definir a forma do documento. Provost [Pro02b] apresenta padrões semelhantes, mas não deixa explícito como a transformação de domínio é realizada e não trata das dificuldades inerentes a esse mapeamento que, como mostra o Capítulo 5, podem tornar sua aplicabilidade bastante difícil. Assim, a forma como esse mapeamento é aplicado é posta à prova, o que resultou no encontro de algumas limitações na transformação de domínio. Entretanto, o ponto de maior interesse nessa pesquisa são os padrões estruturais.

O Capítulo 6 apresenta um catálogo de padrões estruturais. Os padrões estruturais também são relacionados aos padrões para documentos, mas apresentam-se de maneira mais simples, pois constituem apenas um reconhecimento de estruturas de relacionamentos entre elementos UML. Já os padrões para documentos apresentados nesse capítulo são abstrações para o projeto dos vocabulários. Os padrões estruturais serão utilizados no auxílio ao teste de programas que utilizam documentos XML em sua comunicação.

*Tome cuidado com os* bugs *no código acima; eu somente provei sua corretude, mas não o executei.*
***D. Knuth***

# *4 Teste de Software*

Em qualquer processo de desenvolvimento de software, a atividade de testes é fundamental para garantir o sucesso do trabalho. Trata-se de uma atividade bastante custosa, que pode consumir grande parte do esforço para a produção de um sistema [Tas02]. Entretanto a descoberta tardia de defeitos em softwares em ambiente de produção tem um impacto muito maior, causando mais prejuízos e maior esforço posterior para a correção dos defeitos.

O teste de software inclui o projeto dos testes, a execução dos testes e a análise dos resultados obtidos, a fim de se obter maior confiança de que o programa se comporta como desejado. Uma observação importante é que o teste de software não possui ferramentas que possam garantir que um programa está arbitrariamente correto. Esse fato pode ser mostrado observando-se resultados da Teoria da Computação [Pap93]; um resultado bastante famoso sobre o problema da parada diz que dada uma máquina de Turing e uma entrada inicial, determinar se o programa, quando executado com essa entrada, sempre pára (completa) é indecidível. Assim podemos observar que a atividade de testes é um problema bastante desafiador.

Esse capítulo apresenta os principais conceitos relacionados ao teste de software, os quais serão necessários para entender as propostas relacionadas. É apresentada uma visão das técnicas de teste descritas na literatura. Em especial é analisado o método de perturbação de dados, que será estendido com propostas nos próximos capítulos.

## 4.1 Conceitos Importantes

Um dos principais artefatos da atividade de testes é o conjunto de casos de teste. Um caso de teste é formado por um conjunto de valores de entrada para um programa mais a saída esperada. Para determinar se um programa computou um resultado correto geralmente usa-se um oráculo que compara a saída observada com a saída esperada (correta); na maioria dos casos é preciso intervenção humana para se determinar o sucesso da execução de um programa.

Um fato a ser destacado é que o objetivo do teste é revelar defeitos. Assim, um caso de teste bem sucedido é o que revelou um defeito (e não aquele onde se observa o comportamento esperado do programa). Segundo Myers, as seguintes regras ajudam a compreender os objetivos do teste [Mye79]:

a) Na atividade de teste executa-se um programa com o objetivo de descobrir um defeito
b) Os casos de teste devem ter grande probabilidade de revelar um defeito
c) Um teste que revelou um defeito é considerado bem-sucedido

A atividade de teste demonstra que aparentemente os requisitos estão sendo satisfeitos, no entanto ela não pode mostrar a ausência de defeitos (*bugs*). Também se tem uma indicação da confiabilidade e da qualidade do software.

Termos de uso comum na área de testes incluem [IEEE90]: 1) defeito (*fault*): é um problema encontrado em algum artefato do ciclo de desenvolvimento, por exemplo no código fonte ou na especificação; 2) falha (*failure*) é a manifestação concreta de um defeito, pela produção de uma saída incorreta; 3) erro (*error*) é definido como a diferença entre o valor correto e o valor obtido, ou seja, qualquer estado ou resultado diferente do desejado; 4) engano (*mistake*) é uma ação humana que resultou num resultado incorreto.

Uma distinção importante deve ser feita em relação ao objetivo de revelar defeitos. A atividade de teste não se preocupa em remover defeitos, mas somente em encontrá-los. A depuração é o processo que resulta na remoção dos defeitos revelados pelo teste. A depuração deve relacionar um sintoma (erro) a uma causa. Segundo Pressman [Pre01], “A depuração não é teste, mas sempre ocorre como uma conseqüência do teste”. Para realizar a depuração, inicia-se com um caso de teste e sua não-correspondência com o resultado esperado. Então se tenta ligar o sintoma à causa, a fim de se realizar a correção do defeito.

## 4.2 Etapas da Atividade de Teste

A atividade de teste é realizada, em geral, em etapas seqüenciais, que são: teste de unidade, teste de integração, teste de validação e teste de sistema [Pre01]. A seguir são descritas essas etapas.

O teste de unidade tem por objetivo testar a menor unidade do projeto, ou seja, testar cada módulo separadamente. O teste de unidade visa examinar a estrutura de dados local, identificar erros de lógica e de implementação.

A integração busca agregar os módulos que compõem o sistema na estrutura do software. Embora os módulos, depois do teste de unidade, funcionem corretamente de forma isolada, o teste de integração é necessário pois quando os módulos são unidos, situações inesperadas podem acontecer. O teste de integração visa identificar erros associados às interfaces entre os módulos do software.

O teste de validação tem por objetivo testar o programa de acordo com os requisitos funcionais e de desempenho. Ele também é usado para verificar se a documentação está correta e busca verificar se as funções estão de acordo com a especificação.

Durante o teste de sistema são verificados: função e desempenho; tolerância à falhas; segurança; situações anormais do sistema. O teste de sistema considera o software dentro do seu ambiente mais amplo (aspectos de interação com outro hardware, software, pessoas, etc.). Os testes têm por objetivo verificar se todos os elementos do sistema realizam corretamente suas funções.

## 4.3 Técnicas de Teste

As pesquisas para auxiliar a atividade de teste resultaram em técnicas de teste, que em geral são divididas em três grupos: funcionais, estruturais e baseadas em defeitos. Cada técnica possui princípios que guiam os critérios de testes.

Um critério de teste é uma discriminação de possibilidades que servem para julgar questões relativas à atividade de testes [FW88]. O critério deve definir um modo de apreciar os testes, respondendo questões de cunho crítico. Por exemplo, uma questão que surge na realização dos testes é: como saber se o software já foi testado suficientemente (já que é

impraticável testar um programa para todas as entradas possíveis). Para responder a essa pergunta é possível usar métricas e modelos de confiabilidade. Podem-se também ter critérios para a seleção de casos de teste. Nesse caso é possível usar propriedades empíricas que servem como guias para a seleção.

Um critério de adequação define quais propriedades de um programa precisam ser exercitadas [GG75]. O termo domínio de cobertura denota um conjunto de entidades relacionadas (funções, predicados, decisões, etc.) que são verificadas durante a aplicação de algum critério. Dessa maneira é possível encerrar a atividade de testes quando todos os elementos requeridos por um critério de teste forem cobertos.

As técnicas de teste devem ser vistas como complementares umas às outras, pois cada uma exercita diferentes características do software que está sendo testado. Seus objetivos são, dessa maneira, complementares [Mal91]. Os três tipos de técnicas são [Bei90]:

**Técnica Funcional**: utiliza os aspectos funcionais do software para criar os testes. Não se preocupa com a forma como foi implementado o software e é geralmente conhecida como teste de caixa preta. São exemplos da técnica funcional: análise do valor limite, particionamento em classes de equivalência e grafo de causa e efeito [Pre01].

**Técnica estrutural**: leva em conta o funcionamento interno do programa, para verificar se a estrutura interna está de acordo com as especificações, sendo tais testes chamados de teste de caixa branca. Os testes projetados fazem um exame dos detalhes procedimentais, analisando os caminhos lógicos do software. São realizados durante as fases iniciais de desenvolvimento, e buscam revelar: erros lógicos e pressuposições incorretas nos caminhos; caminhos executados de forma errônea; e erros tipográficos aleatórios [Pre01]. Podem-se citar como exemplos de critérios para a técnica estrutural: baseados na complexidade [MB89], baseados em fluxo de controle [GG75] e baseados em fluxo de dados [FW88, Mal91, RW85].

**Técnica Baseada em Defeitos:** escolhe de forma adequada defeitos típicos para criar os casos de teste. A técnica baseada em defeitos usa dados de teste que buscam mostrar a ausência (ou presença) de erros previamente especificados. Por exemplo, para demonstrar que um software trata a divisão por zero corretamente, o dado de teste deve levar à execução dessa divisão por zero. O objetivo é verificar se o programa está livre de defeitos típicos e erros comuns cometidos durante o desenvolvimento. São exemplos da técnica baseada em defeitos: semeadura de erros e Análise de Mutantes [DLS78]. A Análise de Mutantes é uma técnica fundamental para essa pesquisa, pois a perturbação de dados é uma variação dessa técnica.

Análise de Mutantes é uma técnica de testes baseada em defeitos que cria versões modificadas de programas, chamados mutantes. O testador deve encontrar casos de teste que façam o programa mutante se comportar diferentemente do programa em teste. A Análise de Mutantes se baseia em dois pressupostos:

1) Hipótese do programador competente: os programadores escrevem os programas de forma competente, procurando correção. Assim, se um programa estiver incorreto, ele estará próximo do correto;

2) Efeito do acoplamento: Os defeitos complexos em um programa são originados de um encadeamento de defeitos simples. Portanto, se os casos de teste gerados tiverem como objetivo revelar os defeitos simples, temos como conseqüência a revelação de defeitos complexos.

## 4.4 Teste de *Web Sevices*

De acordo com o W3C, um *Web Service* é um sistema de software projetado para proporcionar interoperabilidade na interação máquina-para-máquina sobre uma rede [W3C02]. Sua interface é descrita num formato que pode ser processado pela máquina. Outros sistemas interagem com um *Web Service*, conforme a definição de sua interface, usando mensagens. As mensagens normalmente são encapsuladas num envelope SOAP [W3C03].

SOAP (acrônimo para *Simple Object Access Protocol*) é um protocolo para a troca de mensagens baseadas em XML, que é comumente implementado usando-se HTTP. SOAP constitui-se da camada fundamental para os *Web Services*, provendo um *framework* de troca de mensagens, sobre o qual outras camadas de abstração podem ser construídas. SOAP foi projetado para facilitar o padrão *Service-Oriented Architecture* (SOA) [BBFJS06]. Existem diversos tipos diferentes de padrões de mensagens em SOAP, entretanto o padrão mais comum é o *Remote Procedure Call* (RPC); nesse caso quando um nodo da rede (o cliente) envia uma mensagem de requisição para outro nodo (o servidor), imediatamente é enviada uma mensagem de resposta para o cliente.

Um exemplo de requisição SOAP é mostrado na Listagem 4. 1. O elemento `Envelope` possui atributos sobre as versões utilizadas. O elemento `Body` acomoda a requisição RPC; nesse fragmento, o nome do procedimento remoto é `procedimentoExemplo`, o qual possui um argumento de nome `argumentoExemplo`.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <ns1:procedimentoExemplo xmlns:ns1="http://soapinterop.org/">
      <argumentoExemplo xsi:type="xsd:string">valorArgumento</argumentoExemplo>
    </ns1:procedimentoExemplo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

*Listagem 4. 1 :Exemplo de requisição SOAP*

O modo bastante específico de comunicação dos *Web Services* introduz uma complexidade considerável na atividade de teste, dado que a comunicação sobre uma rede difere em geral dos programas tradicionais, cujas técnicas de teste não podem ser aplicadas de forma direta no domínio de *Web Services*.

O mecanismo de troca de mensagens dos *Web Services* pode ser testado com foco na mensagem SOAP, assim como um teste mais específico pode ser feito no formato da mensagem SOAP [BLO02]. Offut et al [OL01, OW04] descrevem métodos para o teste de mensagens entre componentes, com aplicações para mensagens XML arbitrárias, assim como para mensagens SOAP.

Outra forma de teste pode incluir a interface dos *Web Services*, definida por meio da WSDL (*Web Services Description Language*), uma linguagem que provê meios para descrever os *Web Services* [BLO02]. Ainda, o teste sobre UDDI (*Universal Description, Discovery and Integration*) pode ajudar no planejamento de novas funcionalidades dos *Web Services*. A plataforma UDDI é utilizada para publicar novos serviços e assim pode-se testar se os registros UDDI possibilitam que os consumidores dos *Web Services* se comuniquem corretamente; ou ainda que modificações nos serviços não introduzem novos defeitos nessa comunicação. Entretanto essa abordagem é pouco aplicada.

Uma abordagem importante é a simulação de consumidores e produtores [BLO02]. Ferramentas de teste podem simular os consumidores dos *Web Services* (enviando mensagens para eles), assim como podem simular os produtores (retornando mensagens).

Dentre as técnicas de teste existentes, a mais importante para esse trabalho é a Perturbação de Dados, introduzida por Offutt e Wuzhi em [OW04]. A próxima seção apresenta em detalhes essa técnica de teste.

## 4.5 Perturbação de Dados

Perturbação de dados é um método de teste construído principalmente com fundamentos da técnica baseada em defeitos. Os fundamentos desse método têm origem em idéias propostas em [OL01]. O princípio do método consiste em modificar uma mensagem utilizada na comunicação de componentes, segundo algum critério. A mensagem modificada é então enviada, a fim de se analisar a resposta, de acordo com o comportamento esperado do componente. Uma tecnologia específica à qual esses princípios podem ser aplicados é o teste de *Web Services*.

Offutt e Wuzhi descrevem algumas variações de perturbação de dados, que exploram diversos aspectos a serem testados nos *Web Services* (dado que algumas ferramentas realizam testes de forma bastante incompleta, testando somente a comunicação RPC ou somente aplicando técnicas gerais para a verificação dos valores dos dados). A perturbação do valor dos dados modifica valores dos elementos das mensagens XML, de acordo com os tipos dos dados; a perturbação de comunicação de *Remote Procedure Calls* (RPC) testa o uso dos dados; a perturbação de comunicação de dados testa os relacionamentos e restrições sobre os dados.

### 4.5.1 Perturbação do Valor dos Dados

Esse tipo de perturbação modifica os valores das mensagens, com base em regras definidas de acordo com os tipos de dados; as regras são baseadas em técnicas de teste de valor limite. Os casos de teste são construídos por meio da análise dos tipos de dados definidos no XML *Schema*. Um caso de teste consiste em substituir o valor de um dado por seu valor limite. A Tabela 4. 1 ilustra alguns casos de valores limites.

*Tabela 4. 1: Exemplos de valores limites para tipos de dados simples*

| Tipo de dado | Valor limite | Exemplo |
|---|---|---|
| string | comprimento máximo; comprimento mínimo; letras maiúsculas; letras minúsculas | 00000000000000000000000000<br>JOÃO DA SILVA<br>web service |
| número | valor máximo; valor mínimo; zero | $2^{63}$ -1<br>- $2^{63}$<br>0 |
| booleano | verdadeiro; falso | true, false |

### 4.5.2 Perturbação de Comunicação de *Remote Procedure Calls* (RPCs)

*Remote Procedure Call* (RPC) ou Chamada de Procedimento Remoto é uma técnica para a construção de aplicações cliente-servidor distribuídas. Baseia-se na extensão da noção convencional de chamada de procedimento local, onde o procedimento chamado precisa residir no mesmo espaço de memória do procedimento que faz a chamada. Assim, dois processos podem estar em sistemas diferentes, conectados por uma rede, onde a chamada das funções é realizada de forma transparente para os processos, independentemetne do espaço de memória onde eles residem. As mensagens usadas na comunicação podem usar XML para descrever os valores para os argumentos do procedimento remoto. As mensagens de resposta são os resultados que as funções remotas calculam, com base na extração dos argumentos da mensagem XML usada na chamada do procedimento.

A perturbação proposta por Offutt e Wuzhi é baseada em técnicas de Análise de Mutantes, com algumas modificações [DO91]. A Análise de Mutantes foi usada inicialmente para criar versões modificadas de programas com base em seu código. Offutt estendeu esse conceito, para criar mutações nos valores dos dados em vez de mutações nos programas.

A perturbação de comunicação de RPCs testa o uso dos dados, de acordo com duas categorias: uso normal e uso SQL. O uso normal ocorre no interior dos programas, enquanto o uso SQL ocorre quando os argumentos são usados para consultas em bancos de dados.

Para gerar os mutantes são definidos operadores de mutação. Operadores tradicionais causam pequenas modificações no código fonte. Os operadores definidos por Offutt, para o teste de mensagens XML, são baseados em modificações nos valores das instâncias. Os operadores numéricos são *Divisão*, *Multiplicação*, *Negativo* e *Absoluto*. O operador *Divisão* troca o valor de um elemento XML n por 1/n. O operador *Multiplicação* troca o valor n por n*n. *Negativo* muda o valor n para –n. *Absoluto* troca o valor n por seu valor absoluto |n|.

Quando existem dois argumentos do mesmo tipo, uma mutação que pode ser aplicada é a troca de seus valores, definida pelo operador *Troca*. Também foram definidos operadores baseados em técnicas de injeção SQL. A injeção SQL inclui um comando SQL numa variável do tipo *string* submetida a um servidor. A intenção é armazenar a *string* no banco de dados, a fim de executar o comando SQL, com o objetivo de realizar um acesso não autorizado. O operador *Não Autorizado* modifica o valor de uma *string* para tentar forjar uma mensagem que poderia acessar o banco de dados de forma maliciosa, adicionando o texto “ OR '1'='1' ” à string. O comando SQL abaixo apresenta o operador aplicado às *strings* `turing` e `enigma`:

```
SELECT username
FROM adminuser
WHERE username='turing' OR '1'='1' AND password ='enigma' OR '1'='1'
```

Nesse caso, em vez de selecionar o registro relativo ao usuário `turing`, são selecionados todos os usuários da tabela.

### 4.5.3 Perturbação de Comunicação de Dados

A perturbação de comunicação de dados inclui o teste de troca de dados, onde instâncias de documentos XML são comunicadas entre componentes. A comunicação de dados pode incluir definições baseadas em um vocabulário XML, que restringe a forma das instâncias trocadas entre os componentes.

Offutt e Wuzhi propuseram operadores de perturbação que analisam restrições e relacionamentos. Os operadores que testam restrições atuam sobre os valores permitidos para os tipos de dados, com base na recomendação do W3C para tipos de dados [W3C04.a]. Por exemplo, se para um valor são permitidos no máximo 4 dígitos, um caso de teste pode incluir o número 9999.

Os operadores que testam os relacionamentos são baseados na cardinalidade dos elementos XML (conforme a definição do XML *Schema*, com o uso de `maxOccurs` e `minOccurs`). Existem três estratégias de teste nesse caso:

- se um elemento é opcional, devem existir dois casos de teste: um contém uma instância do elemento, outro contém uma instância vazia

- se um elemento pode aparecer uma ou mais vezes, devem existir dois casos de teste: um contém uma instância do elemento, outro contém um número permitido de instâncias

- se um elemento pode aparecer zero ou mais vezes, devem existir dois casos de teste: um contém uma duplicação do valor de um elemento, outro apaga um dos valores de uma instância com múltiplos elementos

A seguir será mostrado um exemplo da aplicação dessas estratégias. Seja o esquema para um vocabulário que trata livros, mostrado na Listagem 4. 2 e uma instância que obedece ao vocabulário, mostrada na Listagem 4. 3.

```
<?xml version="1.0" ?>
<xs:schema>
  <xs:element name="books">
    <xs:annotation>
      <xs:documentation>XML Schema for Books</xs:documentation>
    </xs:annotation>

    <xs:complexType>
       <xs:sequence>
         <xs:element name="book" maxOccurs="unbounded">
           <xs:complexType>
             <xs:sequence>
               <xs:element name="ISBN" type="xs:string" />
               <xs:element name="price" type="xs:double" />
               <xs:element name="year" type="yearType" />
             </xs:sequence>
           </xs:complexType>
         </xs:element>
       </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="yearType">
    <xs:restriction base="xs:int">
      <xs:totalDigits value="4" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

*Listagem 4. 2: XML Schema para um vocabulário que trata livros*

```
<?xml version="1.0"?>
<books>
  <book>
    <ISBN>0-672-32374-5</ISBN>
    <price>59.99</price>
    <year>2002</year>
  </book>

  <book>
    <ISBN>0-781-44371-2</ISBN>
    <price>69.99</price>
    <year>2003</year>
  </book>
</books>
```

*Listagem 4. 3: Instância do vocabulário que trata livros*

Dois casos de teste são mostrados nas listagens seguintes. A Listagem 4. 4 apresenta uma duplicação de um elemento `<book>` e a Listagem 4. 5 apresenta a exclusão de um elemento.

```
<?xml version="1.0"?>
<books>
  <book>
    <ISBN>0-672-32374-5</ISBN>
    <price>59.99</price>
    <year>2002</year>
  </book>

  <book>
    <ISBN>0-781-44371-2</ISBN>
    <price>69.99</price>
    <year>2003</year>
  </book>

  <book>
    <ISBN>0-781-44371-2</ISBN>
    <price>69.99</price>
    <year>2003</year>
  </book>
</books>
```

*Listagem 4. 4: Caso de teste que duplica um elemento*

```
<?xml version="1.0"?>
<books>
  <book>
    <ISBN>0-672-32374-5</ISBN>
    <price>59.99</price>
    <year>2002</year>
  </book>
</books>
```

*Listagem 4. 5: Caso de teste que exclui um elemento*

### 4.5.4 A ferramenta SMAT-WS - Implementação da Perturbação de Dados

Almeida [Alm06] apresenta uma ferramenta para o teste de *Web Services* que implementa diversos conceitos de perturbação de dados vistos anteriormente, chamada SMAT-WS. Sua base se encontra principalmente nos trabalhos de Offutt et al [OL01, OW04], além da proposta de novos operadores de perturbação.

A extensão dos operadores de perturbação consiste de um grupo de transformações sobre a mensagem SOAP, a partir de mensagens existentes, geralmente fornecidas pelo testador. Essa abordagem demonstrou êxito, já que uma porção considerável dos defeitos foi encontrada com seu uso. O fato das novas mensagens serem geradas a partir de modificações de mensagens fornecidas pelo testador torna a abordagem da ferramenta SMAT-WS bastante parecida com a abordagem explorada nesse trabalho.

A ferramenta é utilizada num estudo de caso, apresentando resultados empíricos sobre a eficiência dos operadores de perturbação. Esses resultados foram obtidos a partir da análise de um sistema de *Web Services* reais, num ambiente de produção.

### 4.5.5 Mutação de Esquemas

Recentemente Offutt, Wuzhi e Luo propuseram uma variação da perturbação de dados [OWL05] que difere dos trabalhos anteriores ao produzir mensagens XML **inválidas**; nessa abordagem são definidos operadores de perturbação que alteram os esquemas. As mensagens inválidas são geradas a partir dos esquemas modificados (os operadores são voltados para a mudança de esquemas, e não de instâncias). Apesar de essa abordagem ser baseada em esquemas, os operadores definem mudanças sintáticas simples, não analisando qualquer aspecto semântico do vocabulário. Como os operadores para a criação de esquemas inválidos não são relevantes para a proposta desse trabalho, seus detalhes não foram analisados nesse capítulo.

## 4.6 Considerações Finais

Esse capítulo abordou aspectos importantes da atividade de teste, apresentando seus conceitos fundamentais. As principais técnicas de teste foram mostradas; a técnica de perturbação de dados, aplicada ao teste de *Web Services,* foi descrita em mais detalhes. Essa técnica tem mostrado grande importância para o teste da comunicação de componentes que se utilizam do protocolo SOAP.

Uma limitação do modo de perturbação de dados definido por Offutt e Wuzhi é o fato de somente serem analisadas informações léxicas e sintáticas do modelo. Podem-se observar algumas características das variações de perturbações apresentadas:

- A perturbação do valor dos dados baseia-se em regras simples de troca de valores por valores limites, baseada somente no tipo dos dados.

- A perturbação de comunicação de *Remote Procedure Calls* se preocupa somente com mutações dos valores numéricos dos argumentos de funções remotas ou com argumentos relacionados a consultas SQL.

- A perturbação de comunicação de dados aproxima-se de uma análise do vocabulário XML, pois utiliza as definições de um XML *Schema*. Essas perturbações analisam: 1) as restrições sobre os valores dos dados, definidas no esquema; e 2) os relacionamentos dos elementos, de acordo com a definição da cardinalidade.

Portanto esses operadores de perturbação não são voltados para uma análise semântica dos vocabulários XML. A perturbação de comunicação de dados se aproxima dessa análise, mas apresenta perturbações bastante limitadas, que exploram muito pouco o poder de expressão do XML *Schema*.

Em vista disso, é apresentada nesse trabalho uma nova abordagem de perturbação de dados que trata da análise semântica das mensagens envolvidas na comunicação dos *Web Services*. Essa proposta é baseada na definição de padrões XML, que são encontrados a partir da análise do vocabulário (definido pelo esquema XML) envolvido na comunicação dos componentes.

Para a análise dos padrões presentes em um vocabulário, são definidos padrões estruturais para XML. Os operadores de perturbação propostos são baseados nesses padrões. A seguir, no Capítulo 5, é mostrado o conceito de padrões XML com uso da UML; são mostrados exemplos de como esse conceito pode ser aplicado para a definição de padrões para o projeto de vocabulários XML.

*Ciência da Computação é a ciência da abstração – criar o modelo certo para um problema e inventar a técnica automatizada apropriada para resolvê-lo.*
***A. Aho e J. Ullman***

# *5 Definição de Padrões XML com Modelos UML*

A definição de padrões XML pode se beneficiar das técnicas de mapeamento para UML, pois a mudança do domínio de um modelo XML para o domínio UML pode propiciar uma melhor análise da estrutura do vocabulário. Para demonstrar essa abordagem, nesse capítulo são mostrados mapeamentos de padrões de projeto para o contexto XML, assim como são discutidas as limitações que surgem com essa proposta.

Inicialmente são discutidos aspectos da definição de padrões XML com uso da UML. Em seguida são mostrados dois padrões de projeto, inspirados nos trabalhos relacionados da comunidade de software orientado a objetos. São analisados mapeamentos dos padrões *Bridge* e *Composite*, que têm origem na obra *Design Patterns* [GHJV95].

## 5.1 Padrões XML com o Uso de Modelos UML

O Capítulo 3 mostrou que uma dificuldade da maneira como os padrões para documentos XML normalmente são definidos é que sua definição é puramente textual. Esse fato ocorre por causa da natureza textual dos modelos XML, onde os benefícios da utilização de um modelo visual nem sempre são percebidos.

Já os padrões para aplicativos XML freqüentemente se utilizam de definições expressas por modelos visuais, com base nos modelos UML da comunidade de software orientado a objetos. Entretanto, os modelos UML utilizados são restritos à descrição dos componentes do aplicativo, não sendo aplicados diretamente aos documentos XML.

Esse trabalho faz uma pequena exploração do uso da UML para auxiliar o desenvolvimento de padrões XML. Entretanto a maior atenção da pesquisa está na definição de padrões estruturais e não em padrões de projetos, pois aqueles serão utilizados explicitamente na

atividade de teste. Os padrões nasceram na comunidade de software orientado a objetos, onde os modelos UML foram usados com sucesso para expressar suas estruturas. O uso dos modelos UML para expressar estruturas de documentos XML pode ser muito promissor para a análise de padrões.

Como foi visto no Capítulo 2, o uso da modelagem de vocabulários XML com modelos UML traz muitas vantagens. Um objetivo desse trabalho é mostrar que tal metodologia pode trazer mais vantagens com o uso das técnicas de padrões, constando-se de mais uma motivação para a utilização da modelagem XML auxiliada pela UML. Entretanto, tais vantagens também são atingidas por dificuldades na aplicação da transformação do domínio UML/XML.

Apesar de ambos os modelos, XML e UML, representarem a mesma informação, no contexto de análise de padrões, o modelo UML possibilita uma melhor exploração. A definição do XML *Schema* é puramente textual, seguindo a forma de uma árvore XML. Para se contornar as limitações dessa representação, a definição do vocabulário utiliza elementos XML com significados específicos, não definidos na própria linguagem XML.

Reconhecer padrões com significado definido é algo muito mais difícil de se fazer se somente a árvore XML for analisada. Isso acontece porque a forma de árvore sempre apresenta o mesmo padrão estrutural, sendo muito limitada. Para revelar alguma informação adicional, é preciso conhecer profundamente o significado dos elementos da árvore, assim como seus relacionamentos. Outro problema da análise de padrões em árvores é que árvores muito semelhantes podem representar padrões muito diferentes.

*"O ponto de vista afeta cada interpretação do que é ou não um padrão. O que é considerado um padrão para uma pessoa, pode ser considerado um bloco de construção primitivo para outra pessoa."* [GHJV95]. Como foi exposto, o termo "padrão" pode ter muitos significados, o que possibilita diversas formas de exploração. Nesse capítulo serão tratados padrões de projeto, com o objetivo de mostrar o uso do conceito de definição de padrões XML com base na UML. No próximo capítulo será analisado um outro grupo de padrões, os padrões estruturais (cf. seção 5.5).

## 5.2 Um Mapeamento de Padrões de Projeto para o Contexto XML

Uma estrutura de projeto que possua aspectos comuns identificados é útil para criar modelos reutilizáveis. Esses modelos são aplicados a problemas específicos, quando forem identificadas similaridades. Assim, a descrição de um padrão de projeto mostra quando um padrão

pode ser aplicado, quais são as restrições, quais as conseqüências obtidas, assim como os *trade-offs*.

Para mostrar uma possível forma que os padrões de projeto para documentos XML com o uso de modelos UML pode possuir, foram mapeados padrões conhecidos, originados do software orientado a objetos. Foram escolhidos padrões do livro *Design Patterns* [GHJV95] para realizar essa exploração. A fim de criar uma correspondência, é preciso fazer uma série de adaptações nos padrões, que refletem as mudanças para o domínio XML. Essas adaptações não são fáceis de serem realizadas diretamente, não sendo possíveis para a maioria dos padrões tradicionais. Um trabalho semelhante foi mostrado em [Pro02.b], como foi apresentado no Capítulo 3.

No mapeamento dos padrões do software orientado a objetos, uma dificuldade que surge é o fato de que os recursos de projeto O.O. do XML *Schema* são bastante limitados. Além disso, conceitos como “herança” têm um mapeamento muito particular, que não possui a mesma funcionalidade apresentada no software orientado a objetos tradicional. Assim, o mapeamento do conceito “interface”, por exemplo, apresenta um significado específico no contexto XML, onde não se tem a mesma funcionalidade do software orientado a objetos (um exemplo simples é imaginar que todos os padrões que utilizam interfaces com métodos abstratos devem ser reinterpretados).

Outra dificuldade é que XML não possui uma natureza comportamental. Dessa maneira, os aspectos comportamentais dos padrões de software orientado a objetos, representados pela funcionalidade dos métodos das classes, não apresentam um mapeamento direto.

Os padrões das estruturas de organização dos elementos tornam-se os mais representativos para o contexto XML. Algumas estruturas de projeto tiveram sucesso nos modelos UML, para software orientado a objetos. Tentar mapeá-las para modelos UML que expressam vocabulários XML pode trazer bons resultados, com a criação de padrões de projeto úteis para o contexto XML.

A seguir é explorada a proposta de dois padrões de projeto. A descrição não pretende esgotar esse assunto, que pode ser explorado com a experiência coletiva no projeto de vocabulários XML. O estudo de padrões sempre deixa um convite para o desenvolvimento de novas propostas, que serão compartilhadas por toda a comunidade.

## 5.3 Padrão *Bridge*

### 5.3.1 Descrição

O padrão *Bridge*, como proposto originalmente, tem a intenção de desacoplar uma abstração de sua implementação, de forma que as duas possam variar independentemente [GHJV95]. Vê-se abaixo como o padrão pode ser adaptado para o contexto de vocabulários XML.

A motivação para o uso do padrão *Bridge* decorre do fato de que nem sempre uma simples organização de conceitos em torno de um relacionamento de herança é suficiente para resolver, de forma flexível, o problema de organizar uma abstração que pode ter diferentes implementações. A maneira mais simples de resolver o problema seria definir uma classe abstrata para a abstração e criar subclasses concretas para as diferentes implementações.

Entretanto esse método acopla muito fortemente a implementação à abstração, o que torna difícil de se modificar, estender e reutilizar abstrações e implementações de forma independente. Para resolver esse problema, o padrão *Bridge* separa em duas hierarquias de classes a abstração e a implementação. Isso desacopla as possíveis implementações das abstrações que irão de fato usar qualquer tipo de implementação.

A forma como o padrão foi definido inicialmente está em função dos aspectos comportamentais das classes. Entretanto, como não faz sentido no contexto de XML considerar o comportamento (funções) das classes, o padrão será adaptado para servir à modelagem do estado (elementos) que as classes modelam.

### 5.3.2 Estrutura

A estrutura do padrão Bridge, adaptada para o contexto de aplicações XML, é definida no modelo UML da Figura 5. 1.

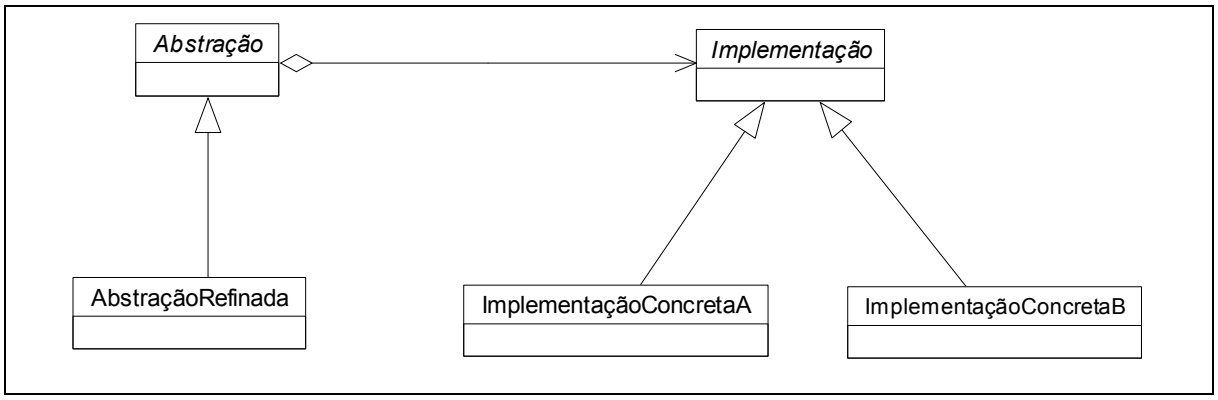



*Figura 5. 1: Estrutura do padrão Bridge para XML*

Nesse modelo, `Abstração` define uma interface e possui uma referência à classe `Implementação`. A `AbstraçãoRefinada` estende a classe `Abstração`, adicionando estados específicos (elementos). `Implementação` é uma classe abstrata (interface), que permite diferentes implementações concretas, as quais terão liberdade na definição real da implementação.

### 5.3.3 Exemplo

Considere a modelagem de um vocabulário XML que usa o conceito de um veículo (um automóvel ou uma motocicleta, por exemplo). Esse conceito estaria sendo usado no contexto de uma loja que vende diversos tipos de veículos. O diagrama UML da Figura 5. 2 ilustra o modelo.

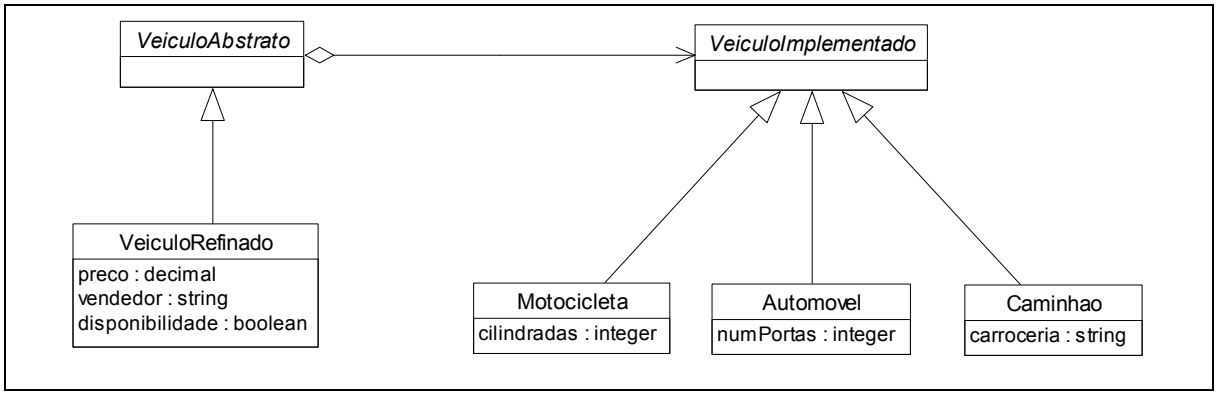



*Figura 5. 2: Modelo UML exemplo para o padrão Bridge*

No modelo, `VeiculoAbstrato` é a interface que é usada para derivar classes que representam veículos refinados, os quais concebem itens de venda da loja. `VeiculoAbstrato` mantém uma ponte (*bridge*) a uma implementação do modelo de veículo, por meio da associação à interface `VeiculoImplementado`. Dessa maneira podemos ter uma flexibilidade na escolha do veículo concreto, como por exemplo com as classes `Motocicleta`, `Automovel` ou `Caminhao`.

O conceito de veículo refinado pode definir elementos específicos (preço, vendedor, disponibilidade), que demonstram como a abstração ficou independente da implementação do conceito de veículo. Desse modo, é possível estender, modificar ou reutilizar essa abstração de forma independente.

Enquanto isso, o conceito de `VeiculoImplementado` ficou fracamente acoplado à abstração, sendo possível obter uma grande flexibilidade na modelagem desse conceito. Podemos imaginar um caso onde, por exemplo, diversos distribuidores modelam seus veículos de forma independente, por meio de um vocabulário compatível com a interface `VeiculoImplementado`. Então as lojas de veículos podem criar suas abstrações de forma totalmente independente das variações possíveis nos diferentes distribuidores de veículos.

A seguir, pode-se observar como o mapeamento do modelo UML desse exemplo é expresso em um esquema XML, o que permite verificar como uma instância que obedece ao vocabulário é criada. A Listagem 5. 1 mostra o *Schema* (simplificado) correspondente:

```
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Automovel  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<xs:complexType name="Automovel">
  <xs:complexContent>
    <xs:extension base="VeiculoImplementado">
      <xs:sequence>
        <xs:element name="numPortas" type="xs:integer"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Caminhao  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<xs:complexType name="Caminhao">...</xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Motocicleta  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<xs:complexType name="Motocicleta">...</xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: VeiculoAbstrato  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<xs:complexType name="VeiculoAbstrato" abstract="true">
  <xs:sequence>
    <xs:element ref="VeiculoImplementado"/>
  </xs:sequence>
</xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: VeiculoImplementado  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<xs:complexType name="VeiculoImplementado" abstract="true"/>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: VeiculoRefinado  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<xs:complexType name="VeiculoRefinado">
  <xs:complexContent>
    <xs:extension base="VeiculoAbstrato">
      <xs:sequence>
        <xs:element name="preco" type="xs:decimal"/>
        <xs:element name="vendedor" type="xs:string"/>
        <xs:element name="disponibilidade" type="xs:boolean"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

*Listagem 5. 1: Esquema correspondente ao modelo da Figura 5. 2*

Na aplicação das regras de mapeamento, note-se como são mapeadas as interfaces, com o uso do atributo `abstract="true"` para as classes abstratas que representam as interfaces de `VeiculoAbstrato` e `VeiculoImplementado`. Uma instância válida que obedece ao vocabulário pode ser definida como:

```
<VeiculoRefinado>
  <Automovel>
    <numPortas>5</numPortas>
  </Automovel>
  <preco>30.000</preco>
  <vendedor>João</vendedor>
  <disponibilidade>true</disponibilidade>
</VeiculoRefinado>
```

Nota-se que a legibilidade da instância XML não é comprometida por qualquer complexidade adicional advinda do uso do padrão *Bridge*. Isso é possível pela maneira transparente como é implementado o mecanismo de herança no XML *Schema* (a instância não precisa conter informações sobre o mecanismo de herança).

## 5.4 Padrão *Composite*

### 5.4.1 Descrição

*Composite* é um padrão usado para compor objetos em estruturas de árvore que representam hierarquias parte-todo [GHJV95]. Com esse padrão, é possível organizar de forma flexível elementos complexos, compostos a partir de elementos mais simples. Podem-se agrupar componentes em componentes arbitrariamente maiores.

O padrão *Composite* usa a composição recursiva para definir o agrupamento dos elementos. Para isso, é usada uma classe abstrata que representa tanto os elementos primitivos, quanto os agrupamentos, que possuem agregações de componentes. No contexto XML, uma classe abstrata servirá para agrupar elementos (atributos) que são comuns às classes primitivas e aos agrupamentos.

### 5.4.2 Estrutura

A estrutura do padrão, no contexto do uso XML, é definida no modelo da Figura 5. 3.

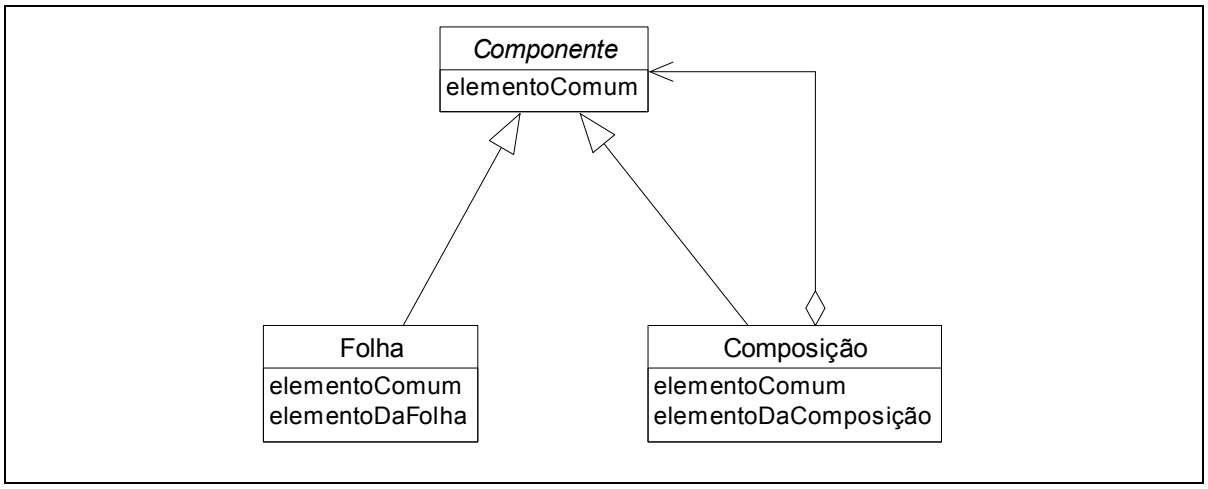



*Figura 5. 3: Estrutura do padrão Composite para XML*

`Componente` é a classe abstrata que define uma interface para as classes `Folha` e `Composição`; a interface define elementos que são comuns a ambas as classes derivadas. `Folha` é a classe que define o fim da hierarquia de composições, representado por um conceito primitivo; ela pode definir elementos (atributos) que são específicos para uma classe que não possui classes filhas (folha). `Composição` define uma agregação de componentes (associação com a classe `Componente`), o que dá origem à composição recursiva; essa classe também pode conter elementos específicos relativos ao conceito de composição.

### 5.4.3 Exemplo

Um bom exemplo de aplicação do padrão *Composite* pode ser visto no vocabulário XML para uma loja que vende coleções de itens semelhantes. Considere uma loja que vende equipamentos de informática. A loja vende itens separados (um mouse, um teclado, um modem) ou vende kits de produtos. Para os elementos vendidos em kits, a loja dá um percentual de desconto. O modelo do vocabulário exemplo é mostrado na Figura 5. 4.

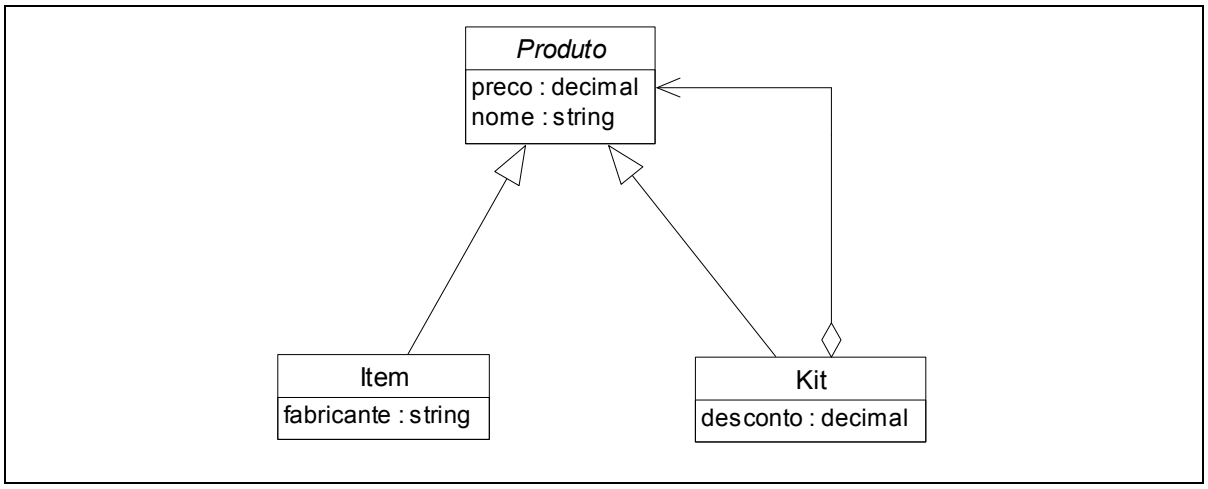



*Figura 5. 4: Vocabulário para loja de equipamentos de informática*

No diagrama, a classe abstrata `Produto` representa os dois tipos de produtos que a loja vende: `Item` (a classe folha) e `Kit` (a composição). A classe `Produto` define dois elementos que são comuns às subclasses, o `preço` e o `nome` do produto. A classe `Kit` representa uma agregação de outros itens ou uma agregação de outros kits; ela possui um elemento específico para esse conceito: `desconto`. A classe `Item` possui o elemento específico `fabricante`.

Um exemplo de instância de kit seria uma agregação dos itens “teclado”, “mouse” e “gabinete”. Já uma instância que agrega o kit anterior e mais um item “monitor” poderia ser chamado de kit “computador”. Dessa maneira, a loja pode organizar diversos kits diferentes, de forma bastante flexível.

Uma vantagem da estrutura desse modelo é que para se descobrir o preço de um kit, não é necessário percorrer toda a árvore. O preço pode ser obtido diretamente a partir da raiz de um kit. Essa propriedade pode ser interessante para aplicações onde percorrer uma grande árvore seja um processo custoso.

A seguir, analisa-se como esse vocabulário é expresso em termos de um esquema XML. A Listagem 5. 2 mostra o XML Schema correspondente.

```
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Item  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<xs:complexType name="Item">
  <xs:complexContent>
    <xs:extension base="Produto">
      <xs:sequence>
        <xs:element name="fabricante" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Kit  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<xs:complexType name="Kit">
  <xs:complexContent>
    <xs:extension base="Produto">
      <xs:sequence>
        <xs:element name="desconto" type="xs:decimal"/>
        <xs:element ref="Produto" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Produto  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<xs:complexType name="Produto" abstract="true">
  <xs:sequence>
    <xs:element name="preco" type="xs:decimal"/>
    <xs:element name="nome" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

*Listagem 5. 2: XML Schema para loja de equipamentos de informática*

Uma instância que obedece ao vocabulário XML é apresentada abaixo. A instância representa um produto composto por um computador e uma impressora; o computador, por sua vez, é composto por outros kits e assim sucessivamente, até os itens primitivos que são mouse, teclado, monitor, etc.

```
<Kit>
  <preco>1454.84</preco>
  <nome>Computador com impressora</nome>
  <desconto>4</desconto>
  <Item>
    <preco>300.00</preco>
    <nome>impressora deskjet</nome>
    <fabricante>HP</fabricante>
  </Item>
  <Kit>
    <preco>1215.46</preco>
    <nome>Computador Padrão</nome>
    <desconto>7.00</desconto>
    <Item>
      ...
      <nome>Monitor SyncMaster</nome>
      ...
    </Item>
    <Kit>
      ...
      <nome>Kit Básico</nome>
      ...
      <Item>
        ...
      </Item>
    </Kit>
  </Kit>
</Kit>
```

Dessa maneira, a loja poderia compor seus produtos de forma bastante natural e flexível, com a aplicação do padrão *Composite* para organizar os elementos de modo conveniente. Uma observação importante é que a complexidade da organização dos produtos (ou outros objetos de outros domínios) é reduzida com a aplicação da recursividade. Um conceito complexo (como um computador formado por inúmeras partes) é reduzido a um elemento raiz, definido em termos mais simples e recursivos.

Um fator que ajuda a simplificar a complexidade é a uniformidade com que são tratados todos os conceitos; isso é possível graças à definição dos conceitos em termos da interface do `Produto` (Componente). Isso é observado na instância XML com o uso dos elementos `preco` e `nome` em ambos os elementos `Item` e `Kit`.

## 5.5 Considerações Finais

Nesse capítulo foi visto o conceito de padrões XML baseados em modelos UML. Uma aplicação desse conceito foi mostrada, para a construção de padrões para projeto de vocabulários XML. Para a definição desses padrões, foi proposto um mapeamento de padrões conhecidos e utilizados no desenvolvimento de software orientado a objetos. Foram mostrados dois exemplos de como essa proposta pode ser aplicada na mudança de domínio de um projeto de vocabulário em

UML para um mapeamento em esquema XML. Também foram mostradas as dificuldades inerentes a essa transformação, onde as diferenças sintáticas entre os modelos em diferentes domínios não têm uma tradução direta. Um exemplo são as limitações da orientação a objetos encontrada nos esquemas XML ou mesmo a deficiência na natureza comportamental da sintaxe dos modelos XML.

Como os padrões de projeto estão relacionados aos padrões tradicionais da comunidade de software orientado a objetos, é necessário ter um contexto específico que permita a aplicação do padrão. O padrão descreve um problema recorrente e sua solução. Esses padrões contêm um conhecimento especializado do domínio do problema, onde são encontradas as melhores práticas e soluções.

Os padrões estruturais, propostos no próximo capítulo, descrevem cenários parecidos com os padrões de projeto, mas são menos específicos e mais simples. Por serem mais simples, sua aplicação não está limitada ao contexto de um problema específico. Nesse sentido, esses padrões lembram os padrões GRASP[3]. Os padrões estruturais serão explorados com o objetivo de se encontrar diretamente a semântica envolvida nos modelos, que se reflete na manifestação dos padrões. No Capítulo 7 os padrões estruturais serão utilizados para a definição de operadores de perturbação de dados.

[3] Padrões GRASP - *General Responsibility Assignment Software Patterns* - são padrões bastante simples, que contêm diretrizes básicas para o projeto de software orientado a objetos. Foram propostos por Larman, na obra Applying UML and Patterns [Lar98].

*Todos os modelos são errados; alguns modelos são úteis.*
***G. Box***

# 6 Padrões Estruturais XML Baseados em Modelos UML

Nesse capítulo são propostos padrões estruturais para documentos XML, cujo objetivo é buscar compreender a semântica da estrutura do diagrama UML envolvido na modelagem XML. Quando o projetista cria um diagrama UML para modelar um vocabulário XML ele irá compor uma estrutura que apresenta um objetivo. Os padrões estruturais podem ajudar a compreender esse objetivo, desvendando a semântica empregada na linguagem UML.

Os padrões estruturais definidos têm um significado diferente dos padrões de projeto analisados no capítulo anterior. O objetivo dos padrões apresentados não é ensinar técnicas complexas de modelagem de vocabulários XML, mas sim explorar o significado de estruturas comumente encontradas nos modelos UML, no contexto do projeto de documentos XML.

## 6.1 Características dos Padrões Estruturais

Uma propriedade dos padrões estruturais propostos é que eles se apresentam de forma bastante simples. Com o uso de padrões simples, existe uma grande possibilidade de encontrá-los num diagrama UML qualquer. Além disso, a análise semântica da estrutura simples pode ser muito bem definida, em relação a uma estrutura complexa que pode apresentar um significado muito específico.

Padrões complexos podem ser criados a partir da construção de modelos que incluem os padrões simples. Os padrões mais específicos podem ser úteis para o projeto de aplicações com alvo bem definido. Esse tipo de padrão está mais relacionado ao conceito de padrão de projeto. Já os padrões simples estão mais próximos do conceito de padrão estrutural.

A descrição de cada padrão que é apresentada busca mostrar o sentido que o padrão tem para a modelagem; também busca indicar quando a estrutura é aplicada. Dessa forma, ao se

reconhecer um padrão num diagrama qualquer, teremos informações semânticas que podem ser exploradas.

Os padrões foram separados em dois grupos: um que trata da estrutura das associações entre as classes; e outro relacionado às estruturas que envolvem o relacionamento de herança. Essa distinção separa melhor os conceitos empregados nos diferentes tipos de relacionamentos estruturais.

## 6.2 Modelos Analisados para a Definição dos Padrões

Os padrões propostos foram inspirados na análise de modelos reais, que apresentam um uso extenso nas aplicações XML. Como a ferramenta usada é o *hyper*Model, os modelos seguem a proposta de exemplos de esquemas de uso comum, descrita no *site* XMLModeling.com, associado à ferramenta. Entre eles estão três modelos de comércio eletrônico de empresas globalmente conhecidas; esquemas de padrões da OASIS[4]; um esquema de padrão da OAGi[5] e um do W3C[6]. A seguir encontra-se uma lista dos esquemas que foram analisados:

1) Serviços Web de Comércio Eletrônico
a) Amazon.com: Serviço de Comércio Eletrônico e Serviço de Informações Alexa
b) eBay: eBay XML API
c) Google: Google Web API

2) Esquemas da OASIS
a) Universal Business Language (UBL), versões 1.0 final e 1.0 beta
b) Security Assertion Markup Language (SAML), versão 1.1
c) Universal Description Discovery and Integration (UDDI), versões 2.0 e 3.0
d) Business Process Execution Language (BPEL), versão 1.1

3) Esquema do Open Application Group (OAGi), versão 8.0

[4] OASIS (*Organization for the Advancement of Structured Information Standards*) é um consórcio global que promove o desenvolvimento, convergência e adoção de padrões de *e-business*

[5] OAGi (*Open Applications Group*) é um grupo que promove padrões para processos baseados em XML, com o objetivo de melhorar a integração de aplicações

[6] W3C (*World Wide Web Consortium*) é um consórcio internacional que desenvolve padrões da *Web*

4) Esquema do W3C - Esquema para Esquemas XML, versão 1.0

Dessa maneira, espera-se que os padrões propostos reflitam a modelagem de aplicações reais e de uso comum por um grande número de desenvolvedores. Na descrição dos padrões, todos os exemplos utilizados são retirados desses modelos, para mostrar a real aplicação dos conceitos.

Os padrões propostos aparecem com uma freqüência específica nos diagramas analisados. Existem padrões que possuem uma freqüência muito alta, assim como padrões que aparecem esporadicamente. Foram analisados 92 diagramas que apresentam padrões de associação e 33 diagramas com padrões de herança. A Tabela 6. 1 mostra a freqüência com que os padrões de associação aparecem; a Tabela 6. 2 mostra a freqüência dos padrões de herança.

*Tabela 6. 1: Freqüência dos padrões de associação, em 92 diagramas*

| **Padrão** | **Freqüência (%)** |
|---|---|
| árvore | 100,0 |
| associação bidirecional | 3,3 |
| associação homóloga | 32,6 |
| associação recursiva | 7,6 |
| atalho | 19,6 |
| ciclo | 1,1 |
| referência múltipla | 40,2 |

*Tabela 6. 2: Freqüência dos padrões de herança, em 33 diagramas*

| **Padrão** | **Freqüência (%)** |
|---|---|
| associação derivada base | 6,1 |
| derivadas homólogas | 48,4 |
| herança multinível | 33,3 |
| herança-associação | 81,8 |

Alguns padrões são bastante comuns, como árvore (100%) ou herança associação (81,8%). Existem também padrões com uma freqüência bastante baixa, como associação bidirecional (3,3%) ou ciclo (1,1%). A maior parte dos padrões possui freqüências com valores médios, que variam de 20% a 50%, aproximadamente.

Como os padrões são baseados em modelos reais, procurou-se prezar pela criação de um catálogo, sem desconsiderar os padrões que possuem pouca freqüência. Vê-se que mesmo os padrões menos populares podem apresentar técnicas de modelagem bastante úteis, que correm o risco de não serem exploradas pelos projetistas.

## 6.3 Padrões de Associação

A organização da descrição de cada padrão é feita da seguinte forma: inicialmente descreve-se a intenção do padrão, de forma sucinta. Em seguida é dada uma descrição detalhada, levando em conta os aspectos descritos anteriormente. Então é apresentada a estrutura do padrão, com um modelo UML, seguida de uma descrição dos elementos participantes da estrutura. Por fim é apresentado um exemplo concreto.

Os padrões de associação expressam a estrutura dos relacionamentos entre os elementos XML, que irá se refletir na organização da árvore que representa uma instância. A seguir são apresentados os padrões de associação.

### 6.3.1 Referência Múltipla

**Intenção**

Modelar um elemento que pode ser referenciado por múltiplas classes como uma classe singular.

**Descrição**

Considere um elemento de um modelo que apresenta características úteis para diversos outros elementos. Uma boa prática de projeto é fatorar essas características reutilizáveis em um conceito, que poderá ser referenciado por outros elementos.

Cada uso do conceito singular está relacionado a um contexto específico. O contexto é representado pelo nome das associações direcionadas ao conceito comum. Deve-se lembrar que no mapeamento do modelo UML para XML tanto os conceitos (classes), quanto as associações darão origem a  elementos. As características reutilizadas, da classe singular, serão elementos filhos de um elemento que tem o nome da associação.

**Estrutura**

A estrutura do padrão é mostrada na Figura 6. 1.

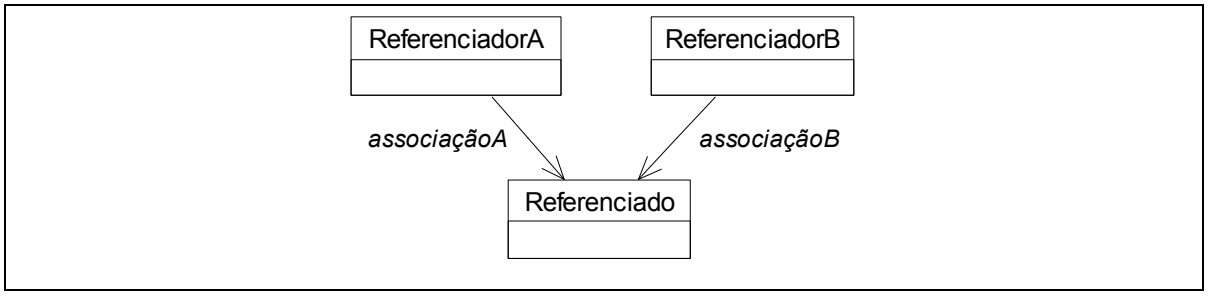



*Figura 6. 1: Estrutura do padrão Referência Múltipla*

**Participantes**

- *Referenciadores*: possuem associações direcionadas a uma classe em comum
- *Referenciado*: a classe singular, à qual as associações se referem

**Exemplo**

O diagrama de classes da Figura 6. 2 mostra o mapeamento UML para o esquema XML usado pelo sistema de comércio eletrônico da Amazon.

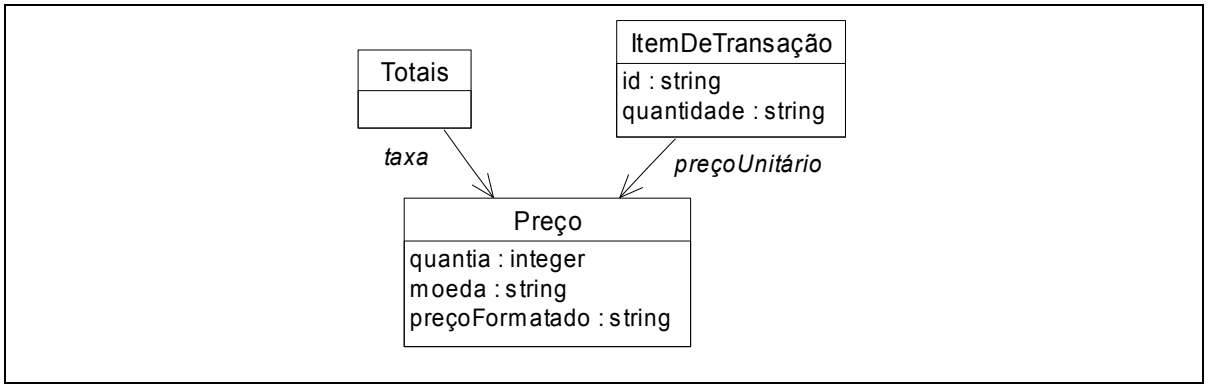



*Figura 6. 2: Padrão encontrado no modelo do Amazon E-Commerce Service*

Nesse modelo destaca-se a classe singular `Preço`, que é utilizada por outras classes: `Totais` e `ItemDeTransação`. Cada uso (associação) do conceito de preço aparece segundo um contexto específico; o preço pode ser o preço unitário de um item ou pode ser o preço das taxas envolvidas na transação. Da mesma forma, o conceito de preço poderia ser usado nos contextos de totais, sub-totais, promoções, entre outros. O esquema XML correspondente é mostrado na

Listagem 6. 1 (caracteres especiais foram substituídos, devido a limitações do editor de XML). Na listagem podem-se observar as particularidades do mapeamento UML/XML.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: ItemDeTransacao  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
  <xs:element name="ItemDeTransacao" type="ItemDeTransacao" />

  <xs:complexType name="ItemDeTransacao">
    <xs:sequence>
      <xs:element name="id" type="xs:string" />
      <xs:element name="quantidade" type="xs:string" />
      <xs:element      name="precoUnitario"      type="Preco"      minOccurs="0"
maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Preco  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
  <xs:element name="Preco" type="Preco" />

  <xs:complexType name="Preco">
    <xs:sequence>
      <xs:element name="quantia" type="xs:integer" />
      <xs:element name="moeda" type="xs:string" />
      <xs:element name="precoFormatado" type="xs:string" />
    </xs:sequence>
  </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Totais  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
  <xs:element name="Totais" type="Totais" />

  <xs:complexType name="Totais">
    <xs:sequence>
      <xs:element name="taxa" type="Preco" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

*Listagem 6. 1: Esquema XML correspondente ao modelo da Figura 6. 2*

Devido a restrições de espaço, nesse capítulo não serão mostrados os XML *Schemas* correspondentes aos modelos UML nos próximos padrões. Entretanto os esquemas correspondentes encontram-se no Apêndice A, caso o leitor deseje analisar o mapeamento. O mapeamento é obtido de forma direta, aplicando-se as regras vistas no Capítulo 2.

### 6.3.2 Associações Homólogas

**Intenção**

Expressar a associação entre dois conceitos em diferentes contextos. Cada contexto dá um sentido diferente a cada uma das associações “homólogas”.

**Descrição**

As associações entre duas classes (no modelo UML) mostram o relacionamento entre dois elementos (no modelo XML), que têm uma semântica definida pelo projetista, expressada pelo nome da associação. Entretanto, pode existir mais de um relacionamento entre esses elementos, o que exige mais de uma associação entre as classes, cada uma para expressar um relacionamento com sentido específico.

Podem existir assim várias associações homólogas entre dois conceitos do diagrama de classes, cada uma expressando uma semântica diferente. Dessa maneira, essa estrutura mostra que os elementos podem se relacionar em diferentes contextos nos documentos XML. Esse padrão mostrou-se de uso bastante freqüente em diversos modelos analisados.

**Estrutura**

A estrutura do padrão é mostrada na Figura 6. 3.

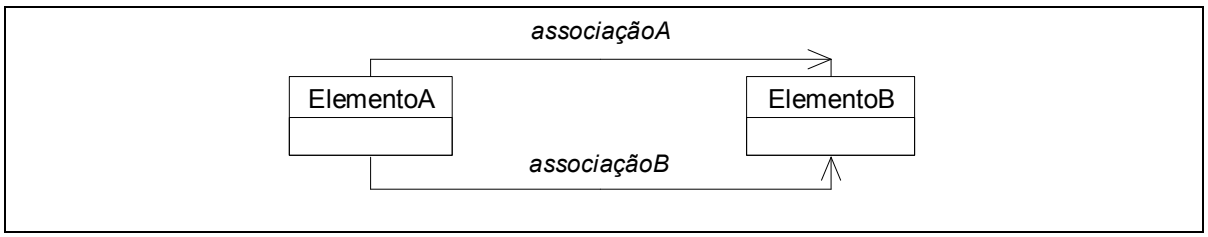



*Figura 6. 3: Estrutura do padrão Associações Homólogas*

**Participantes**

- *Elementos*: conceitos que possuem múltiplos relacionamentos
- *Associações*: mostram os contextos em que os elementos se relacionam

**Exemplo**

O diagrama de classes da Figura 6. 4 mostra o mapeamento UML para o esquema XML usado pelo sistema de comércio eletrônico da Amazon.

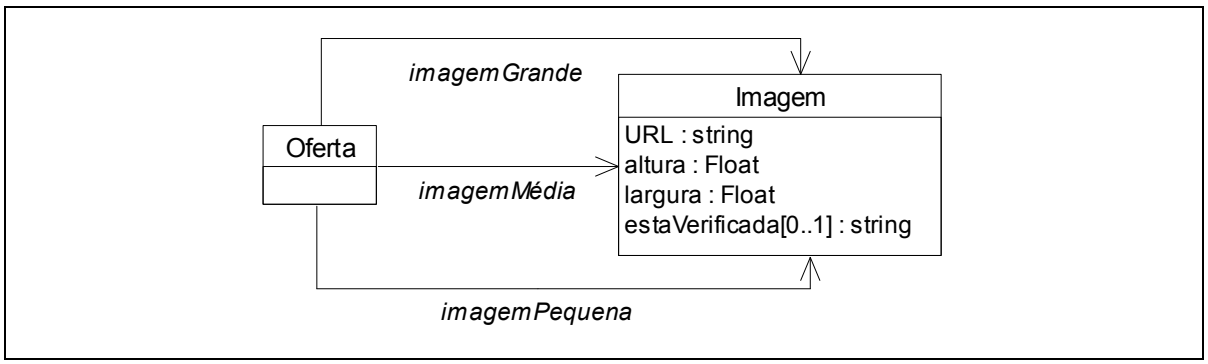



*Figura 6. 4: Padrão encontrado no modelo do Amazon E-Commerce Service*

Esse exemplo mostra três contextos diferentes para associações entre os elementos `Oferta` e sua respectiva `Imagem`. Um produto em oferta pode ser relacionado a imagens pequenas, médias e grandes, dependendo das opções do usuário do portal Amazon. Deve-se destacar o fato de que o que dá sentido ao relacionamento é o nome da associação (que pode ser até mesmo omitido em modelos sem ambiguidade).

### 6.3.3 Árvore

**Intenção**

Modelar uma hierarquia de elementos em forma de árvore, onde os elementos filhos constituem sub-árvores.

**Descrição**

O padrão mais comum encontrado nos esquemas XML é o padrão de árvore. Uma árvore é o mapeamento mais direto para a estrutura de um documento XML, pois sua organização natural é composta de elementos (raízes de árvores) que contêm sub-elementos (sub-árvores ou folhas).

Para o fim de análise do padrão é interessante verificar a forma da árvore. A posição de uma classe na árvore pode revelar a semântica envolvida nas associações entre as classes no

modelo UML. Assim, o relacionamento entre classes (nodos) irmãs tem um significado próprio; enquanto o relacionamento entre classes onde, por exemplo, na hierarquia uma é neta da outra tem outro sentido a ser analisado.

Pode-se pensar em muitas outras características do padrão árvore, como por exemplo, o papel desempenhado pelas classes folhas, em relação ao papel das classes não-folhas. Essa análise depende do uso do padrão.

**Estrutura**

A estrutura do padrão é mostrada na Figura 6. 5.

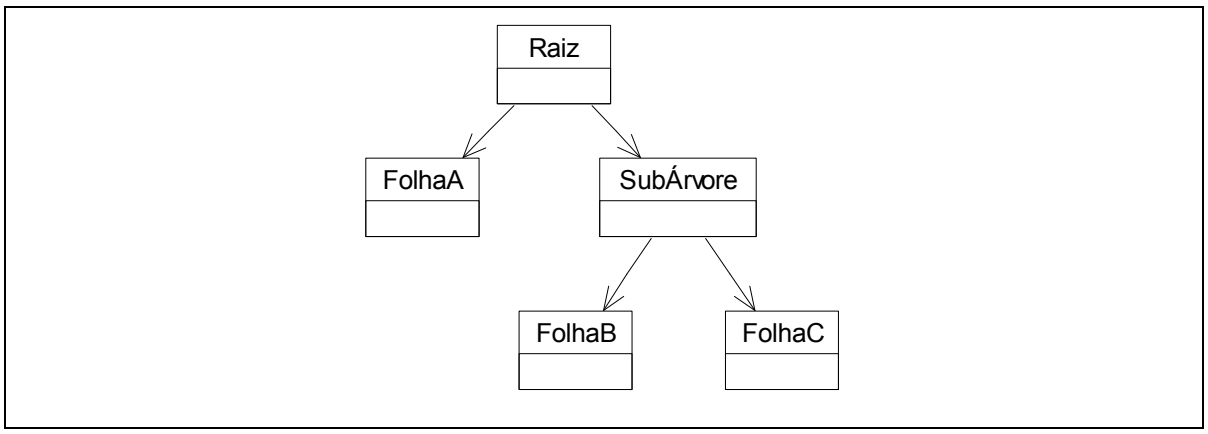



*Figura 6. 5: Estrutura do padrão Árvore*

**Participantes**

Os participantes são os elementos encontrados comumente nas estruturas de árvore, como folhas, raízes, sub-árvores, etc.

**Exemplo**

O diagrama de classes da Figura 6. 6 mostra o mapeamento UML para o esquema XML usado pelo sistema de comércio eletrônico da Amazon.

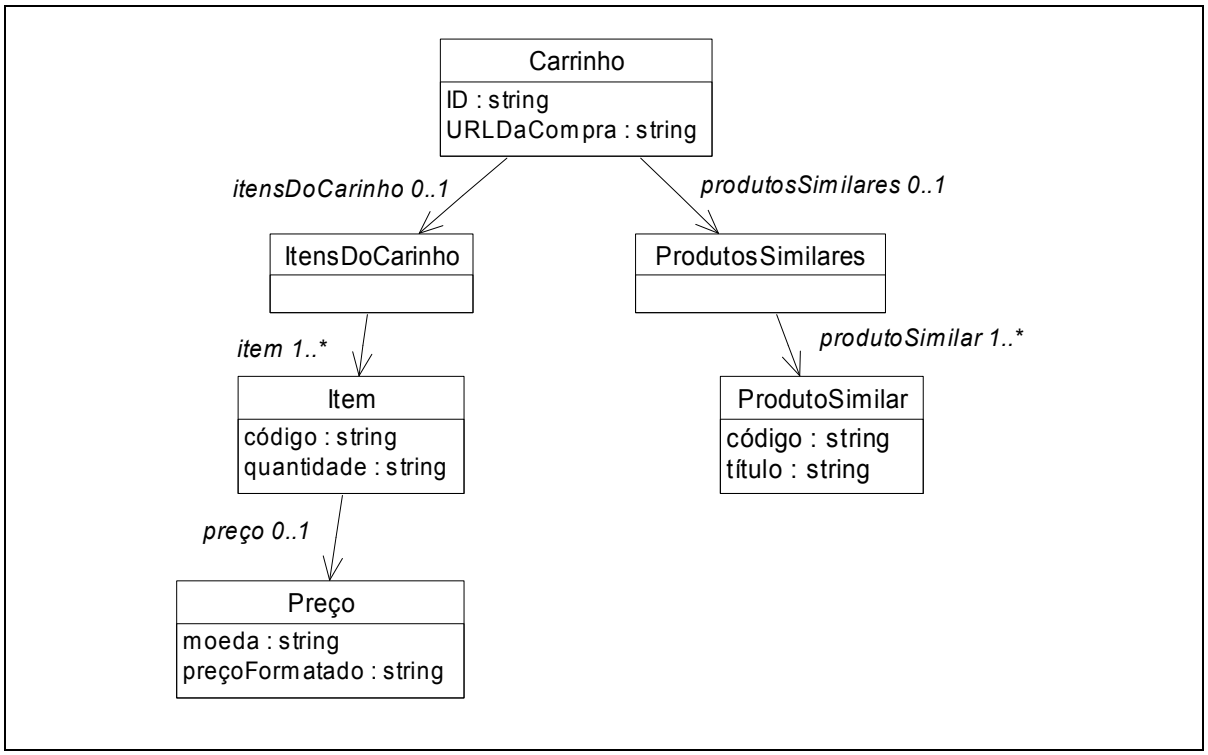



*Figura 6. 6: Padrão encontrado no modelo do Amazon E-Commerce Service*

Esse exemplo mostra como a estrutura de árvore foi usada para projetar a estrutura hierárquica do carrinho de compras da loja virtual. Notamos que as classes irmãs tendem a representar conceitos bastante relacionados. Já as classes com um maior nível de indireção (associações intermediárias) na navegabilidade refletem a organização natural em forma de hierarquia entre os conceitos.

### 6.3.4 Associação Recursiva

**Intenção**

Expressar um relacionamento entre elementos, definido de forma recursiva. Cada elemento pode ser relacionado (associado) a sub-elementos, definidos com base na mesma estrutura desse elemento.

**Descrição**

Muitas vezes existem conceitos que se relacionam a outros conceitos, sendo que todos eles são definidos nos mesmos termos. Esse tipo de estrutura dá origem a uma organização

hierárquica em forma de árvore entre esses conceitos. A organização é expressa por elementos que possuem sub-elementos de seu próprio tipo.

Esse padrão pode ser visto como uma expressão do modelo de um conjunto dividido em subconjuntos. Qualquer conjunto (elemento) nessa hierarquia pode conter um número arbitrário de subconjuntos (elementos do mesmo tipo do elemento raiz). Portanto esse padrão é útil sempre que se quer organizar hierarquicamente um grupo de elementos do mesmo tipo.

**Estrutura**

A estrutura do padrão é mostrada na Figura 6. 7.

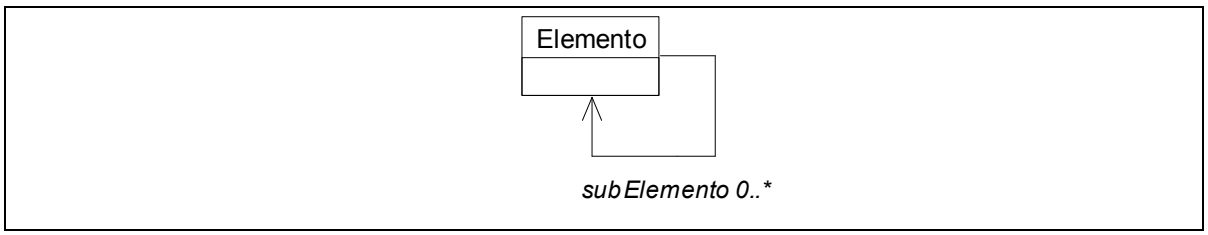



*Figura 6. 7: Estrutura do padrão Associação Recursiva*

**Participantes**

- *Elemento*: compreende a definição de um conceito, que possui relacionamentos com conceitos do mesmo tipo em um conjunto

- *Associação subElemento*: define qual a relação (hierarquia) entre os conceitos do conjunto

**Exemplo**

O diagrama de classes da Figura 6. 8 mostra o mapeamento UML para o esquema XML usado pelo sistema de comércio eletrônico da eBay.

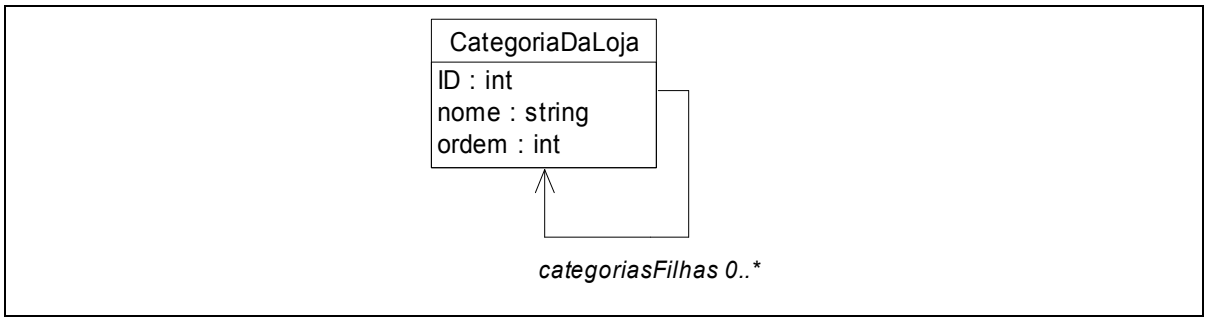



*Figura 6. 8: Padrão encontrado no modelo do eBay*

É comum que os *sites* de compras organizem os produtos disponíveis em categorias. Pode-se ter, por exemplo, um conjunto que inclui a categoria “Computadores”, que inclui a subcategoria “Componentes”, que possui o subconjunto “Monitores”. Essa estrutura pode ser representada pelo padrão de forma simples, de maneira como foi feita a organização das categorias da eBay em categorias filhas.

### 6.3.5 Atalho

**Intenção**

Referenciar um mesmo conceito a partir de diferentes níveis de indireção na navegabilidade das associações.

**Descrição**

O padrão pode ser visto como um caso especial do padrão Referência Múltipla, onde uma das classes que possui referência para o conceito singular possui navegabilidade para esse conceito por meio de associações que levam a uma segunda classe que possui referência para o conceito singular. Dessa forma uma das duas navegabilidades terá um nível de indireção menor.

Nota-se que o padrão Atalho sempre apresentará também o padrão Referência Múltipla. Entretanto a classe que possui diferentes níveis de indireção para o conceito singular merece maior atenção, dado que a sintaxe da estrutura permite a instanciação de árvores XML onde essa classe poderá apresentar associação para o conceito, ao mesmo tempo em que o conceito pode estar presente em diferentes níveis de sua sub-árvore. A semântica envolvida na escolha do nível desse conceito na árvore pode ser de interesse para aplicações que forem se utilizar da análise de padrões.

**Estrutura**

A estrutura do padrão é mostrada na Figura 6. 9.

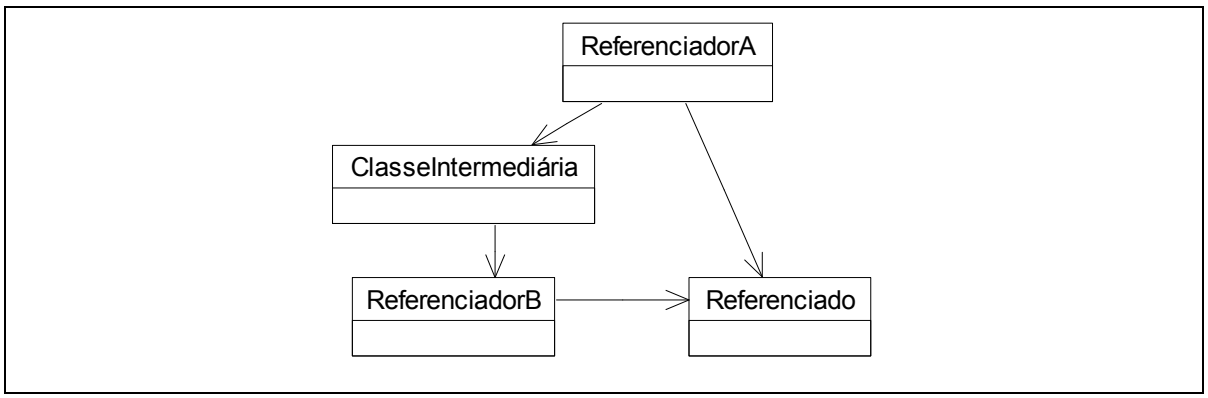



*Figura 6. 9: Estrutura do padrão Atalho*

**Participantes**

- *ReferenciadorA*: classe que possui navegabilidade para o conceito singular por diferentes níveis de indireção, onde um dos níveis permite a navegação para a classe ReferenciadorB; e o outro é uma associação direta
- *ReferenciadorB*: classe que possui navegabilidade direta para o conceito singular
- *Referenciado*: o conceito singular
- *Classe intermediária*: zero ou mais classes podem ser intermediárias, o que cria a indireção. No caso mais simples, ReferenciadorB é a única classe intermediária

**Exemplo**

O diagrama de classes da Figura 6. 10 mostra o mapeamento UML para o esquema XML usado pela Amazon Alexa Web Information Service.

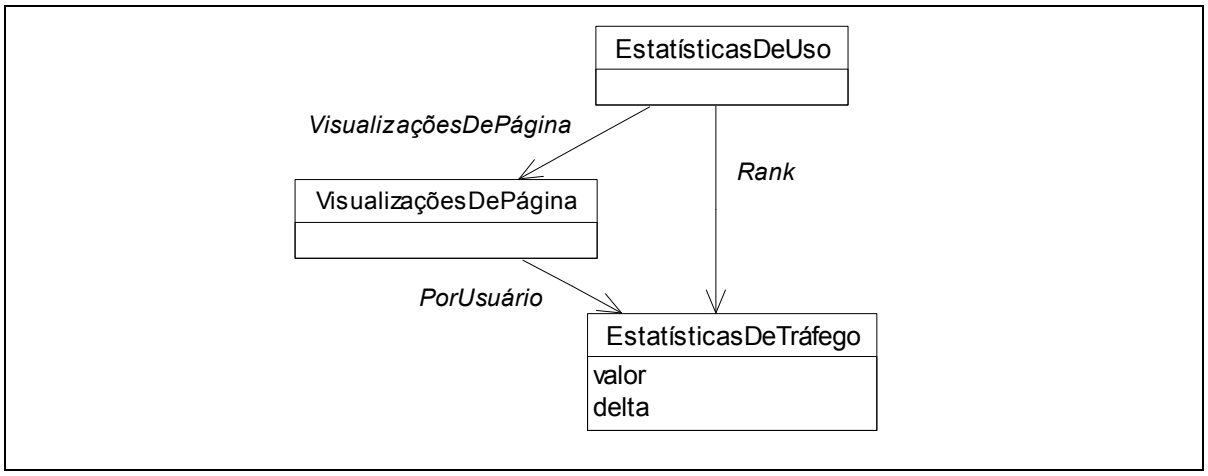



*Figura 6. 10: Padrão encontrado no modelo da Amazon Alexa Web Information Service*

Nesse exemplo o conceito `EstatísticasDeTráfego` tem navegabilidade a partir da classe `EstatísticasDeUso`, por meio de ambas as associações `VisualizaçõesDePágina` e `Rank`. Nota-se que o nome da associação (`Rank` ou `PorUsuário`) determina a semântica da estatística de tráfego: esse conceito singular poderá estar presente em diferentes níveis da árvore XML, com significados diferentes.

### 6.3.6 Associação Bidirecional

**Intenção**

Criar um relacionamento entre dois elementos, onde o primeiro elemento referencia o segundo e vice-versa.

**Descrição**

A maioria das associações determina uma única direção para esse tipo de relacionamento. Entretanto um caso especial ocorre quando a associação deve ser nos dois sentidos. Nessa situação, devem-se criar duas associações, que serão mapeadas para dois elementos XML.

Essa construção possibilita que possam ser criadas árvores de elementos XML onde os tipos de elementos se alternam a cada nível. Trata-se de uma estrutura interessante para a análise do padrão, pois, a cada instância, a posição de um elemento poderá ser alternada para qualquer nível da árvore, obedecendo à restrição de alternância de tipos entre níveis.

Uma propriedade interessante a ser observada é que o padrão possibilita que dois conceitos estejam associados com uma visibilidade total de um para outro. Esse tipo de visibilidade pode ser interessante quando tem-se duas instâncias de documentos XML, onde em uma o primeiro elemento é raiz do segundo; e na outra ocorre o contrário. Assim, não importa se um ou outro elemento é a raiz da árvore: os dois conceitos podem estar associados em ambos os casos.

**Estrutura**

A estrutura do padrão é mostrada na Figura 6. 11.

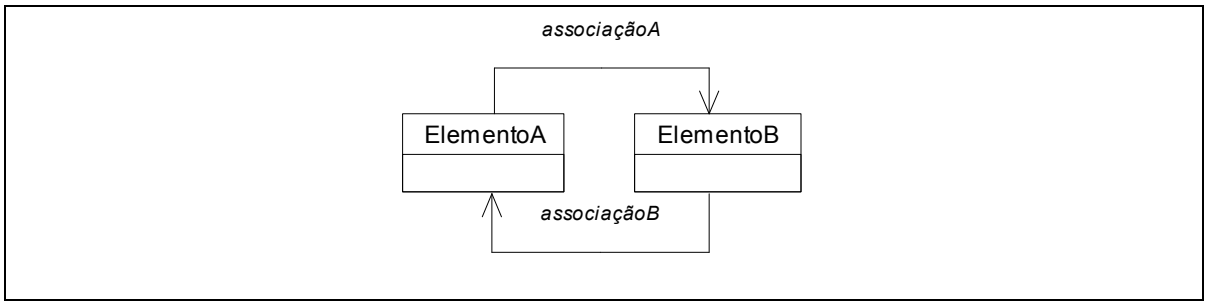



*Figura 6. 11: Estrutura do padrão Associação Bidirecional*

**Participantes**

- *Elementos*: possuem associações bidirecionais
- *Associações*: o par de associações bidirecionais de dois elementos

**Exemplo**

O diagrama de classes da Figura 6. 12 mostra o mapeamento UML para o esquema XML usado pela OASIS Universal Business Language (UBL).

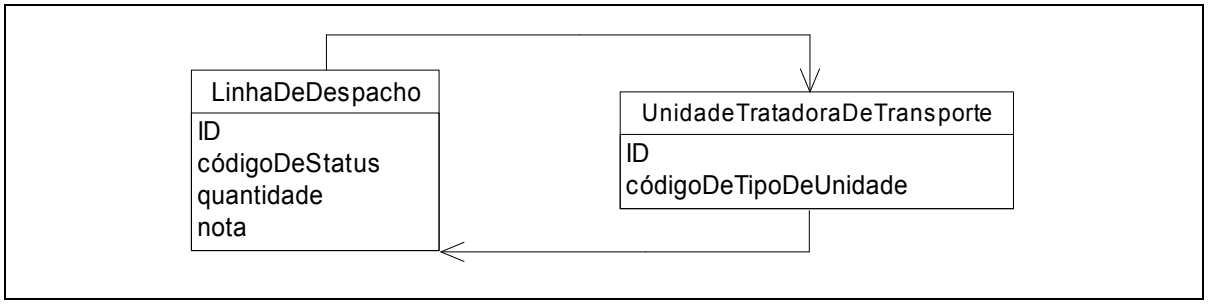



*Figura 6. 12: Padrão encontrado no modelo da OASIS Universal Business Language (UBL)*

Esse exemplo mostra uma aplicação da estrutura do padrão, onde uma linha de despacho pode estar associada a uma unidade tratadora de transporte, e vice versa. Os projetistas desse vocabulário identificaram que a visibilidade nos dois sentidos é útil, o que pode ser importante em diversas situações.

### 6.3.7 Ciclo

**Intenção**

Criar um ciclo fechado de associações, onde não há necessariamente uma classe raiz da hierarquia.

**Descrição**

O padrão Ciclo ocorre quando a estrutura de associações permite que a partir de uma classe se possa voltar a essa mesma classe pela navegação das associações; assim não há uma classe raiz definida. Nesse caso temos uma extensão do padrão Árvore, pois um ciclo contém uma árvore.

Esse tipo de organização possui uma característica interessante: as instâncias de árvores XML possuem uma profundidade não especificada e cíclica. Um conjunto de associações que formam um ciclo proporciona um grau de liberdade bastante alto na instanciação de documentos XML. Isso pode ser observado pelo fato de que qualquer classe pode assumir o papel de raiz; e além disso todas as outras classes do ciclo podem ser atingidas pela navegação das associações, a partir de qualquer raiz escolhida.

**Estrutura**

A estrutura do padrão é mostrada na Figura 6. 13.

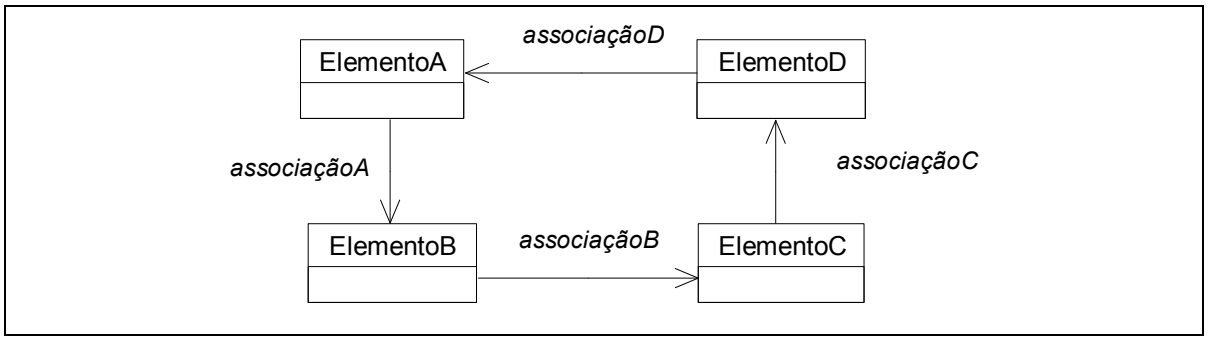



*Figura 6. 13: Estrutura do padrão Ciclo*

**Participantes**

- *Elementos*: classes que fazem parte do ciclo
- *Associações*: definem a estrutura do ciclo

**Exemplo**

O diagrama de classes da Figura 6. 14 mostra o mapeamento UML para o esquema XML usado pelo OASIS Security Services (SAML).

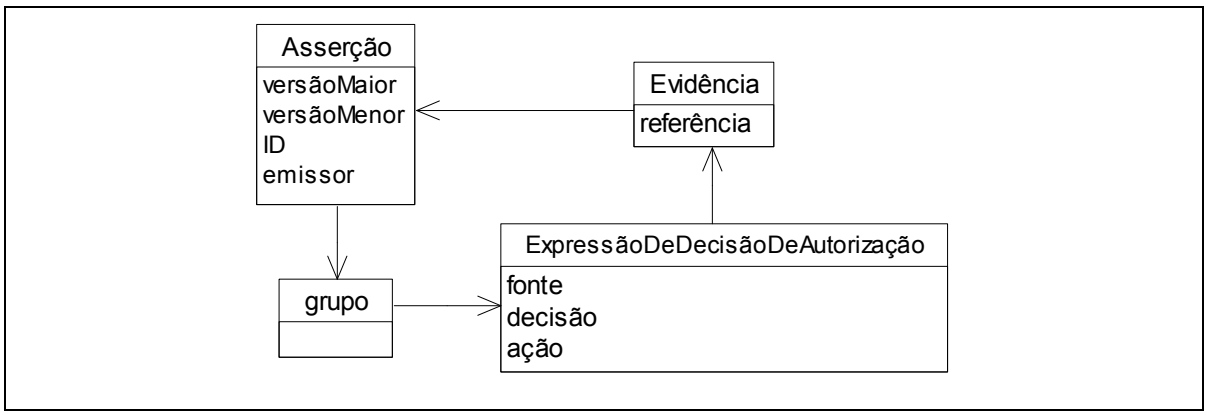



*Figura 6. 14: Padrão encontrado no modelo do OASIS Security Services (SAML)*

Nesse exemplo, observa-se que a estrutura do modelo permite uma instância em que qualquer um dos elementos possa conter como sub-elemento um conceito de seu mesmo tipo, em alguma posição na árvore (caso em que a instância XML apresenta uma volta completa no ciclo). Uma estrutura definida nesses termos, que pode ser vista como uma organização recursiva, pode ser útil para a modelagem dos relacionamentos do vocabulário.

## 6.4 Padrões de Herança

Os padrões de herança aparecem de forma mais específica nos modelos, pois tratam casos especiais onde o relacionamento de herança foi identificado como útil à modelagem. Esse tipo de relacionamento não é muito comum na modelagem XML, pelo fato de que a maioria dos relacionamentos entre as estruturas XML é capturada pelos padrões de associação, que expressam a forma mais comum de relacionamentos nas árvores de elementos das instâncias. A seguir são apresentados os padrões de herança.

### 6.4.1 Herança-Associação

**Intenção**

Estabelecer uma associação específica entre as classes base e/ou derivada com outras classes.

**Descrição**

A partir de uma estrutura simples de herança, é importante, no contexto da modelagem XML/UML, entender que tipo de associação outras classes possuem com a estrutura que envolve a herança. Tanto a classe base quanto a classe derivada podem ter associações com outras classes, onde a navegabilidade pode estar em diversos sentidos (das classes da estrutura de herança para outras, e vice versa).

Nesse trabalho o objetivo não é encontrar todos os padrões possíveis, mas sim os padrões mais significativos, que serão aplicados com objetivos específicos (o auxílio à atividade de testes). Dessa forma, a estrutura básica identificada no estudo dos modelos apresenta a associação entre as classes base, derivada e outra classe externa.

Para entender o objetivo da inserção desse padrão na construção do modelo, é importante observar a associação da classe externa com as classes base/derivada. Por exemplo, se a navegabilidade ocorrer da classe externa para a derivada, a classe externa estará indiretamente referenciando atributos relacionados com a classe base.

**Estrutura**

Uma possível estrutura desse padrão é mostrada na Figura 6. 15. A bidirecionalidade das associações indica que os dois sentidos podem ocorrer, não necessariamente devendo ocorrer.

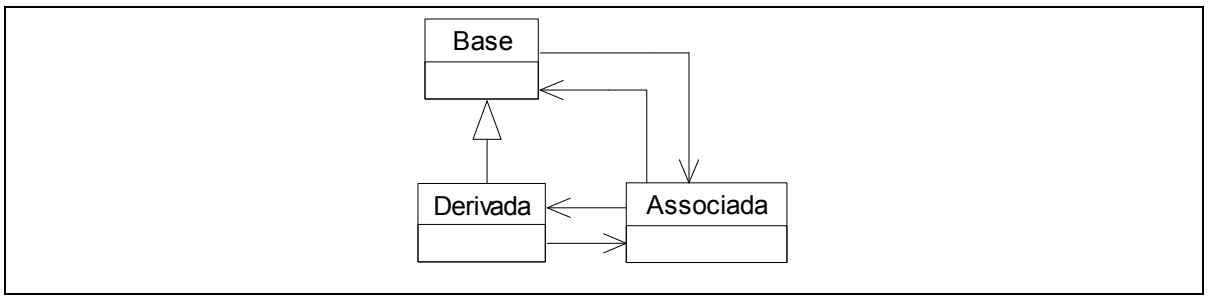



*Figura 6. 15: Estrutura geral do padrão Herança-Associação*

Podemos ter sub-padrões, gerados a partir de simplificações do padrão mais geral. Esses são mais comuns nos modelos. Os padrões mais simples têm uma das formas ilustradas na Figura 6. 16.

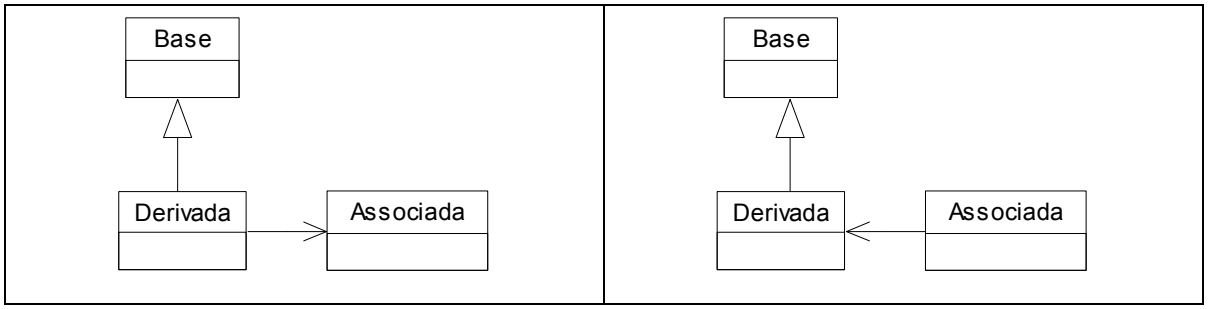



*Figura 6. 16: Sub-padrões Herança-Associação*

Nesse caso é importante observar a semântica envolvida na direção da navegação da associação: ou a classe derivada pode ser referenciada pela classe externa, ou referenciá-la; o caso onde ambas as referencias ocorrem é definido pelo padrão Associação Bidirecional.

Outros padrões mais complexos podem ter a forma representada na Figura 6. 17.

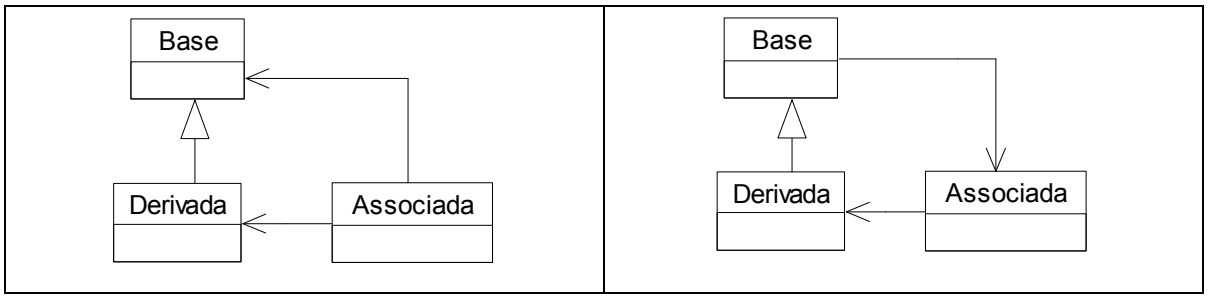



*Figura 6. 17: Sub-padrões Herança-Associação*

No primeiro caso a classe `Associada` referencia tanto a classe `Base`, quanto a `Derivada`, o que dá um grau de liberdade maior para as associações, em relação ao caso anterior (onde a classe `Associada` não referencia a classe `Base`). Já o segundo caso tem uma estrutura onde a classe `Base` referencia a classe `Associada`, que por sua vez referencia a classe `Derivada`. Essa estrutura permite a construção de instâncias XML mais complexas, onde o elemento do tipo da classe `Base` pode conter sub-elementos do tipo da classe `Associada`, que por sua vez pode conter elementos que são especializações do elemento do tipo da classe `Base`.

**Participantes**

- *Base*: classe base que pode referenciar e/ou ser referenciada por uma classe Associada
- *Derivada*: classe derivada que pode referenciar e/ou ser referenciada por uma classe Associada
- *Associada*: representa o potencial de associação com estrutura de herança

**Exemplo**

O diagrama de classes da Figura 6. 18 mostra o mapeamento UML para o esquema XML usado pela OASIS Business Process Execution Language (BPEL).

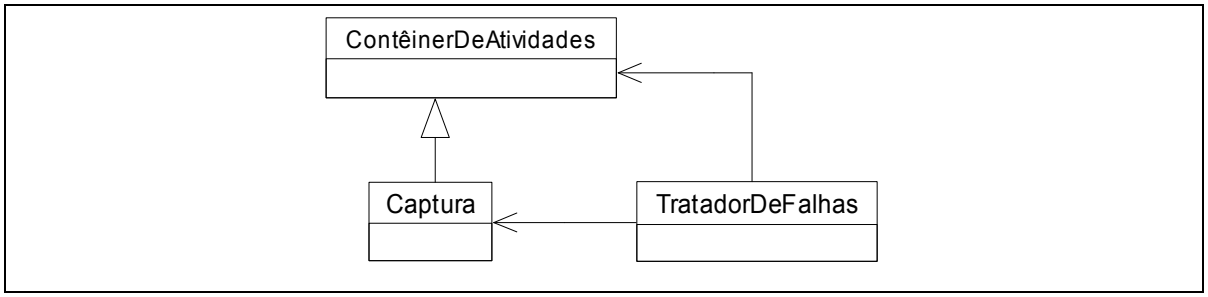



*Figura 6. 18: Padrão encontrado no modelo da OASIS Business Process Execution Language*

Nesse exemplo temos uma estrutura onde um `TratadorDeFalhas` pode conter elementos do tipo `ContêinerDeAtividades`, assim como elementos do tipo `Captura`, uma especialização do tipo contêiner.

A seguir, na Figura 6. 19, temos uma parte da estrutura do modelo da Amazon Alexa Web Information Service.

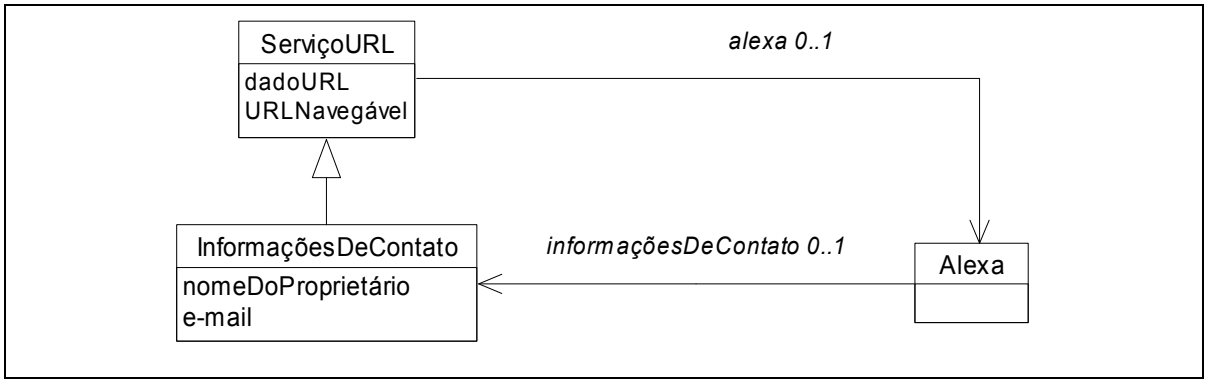



*Figura 6. 19: Padrão encontrado no modelo da Amazon Alexa Web Information Service*

Esse exemplo mostra uma construção interessante: um `ServiçoURL` pode conter um elemento do tipo `Alexa`, que por sua vez pode conter informações de contato, que são representadas por uma classe derivada da classe `ServiçoURL`.

### 6.4.2 Derivadas Homólogas

**Intenção**

Estabelecer um grupo de classes derivadas de uma mesma classe base, as quais podem apresentar relacionamentos entre si.

**Descrição**

Um grupo de duas ou mais classes derivadas é um padrão muito comum em estruturas que envolvem herança. A detecção desse padrão será útil para o fim de elaboração de dados de teste significativos. Quando há associações entre as classes derivadas temos uma semântica que deve ser analisada de maneira especial. Essa associação é diferente da associação vista no padrão Herança-Associação, pois envolve as próprias classes derivadas, e não classes externas.

Vale a pena observar que podemos combinar esse padrão com outros padrões, para obter estruturas mais complexas. Por exemplo, pode-se combinar o padrão com Herança-Associação ou com Referência Múltipla. Entretanto essas combinações não serão tratadas como um novo padrão estrutural, mas sim como uma combinação entre dois ou mais padrões simples. Observa-se que os padrões de projeto podem apresentar os padrões estruturais simples, mas com uma semântica muito mais específica.

**Estrutura**

A estrutura mais geral desse padrão tem a forma mostrada na Figura 6. 20.

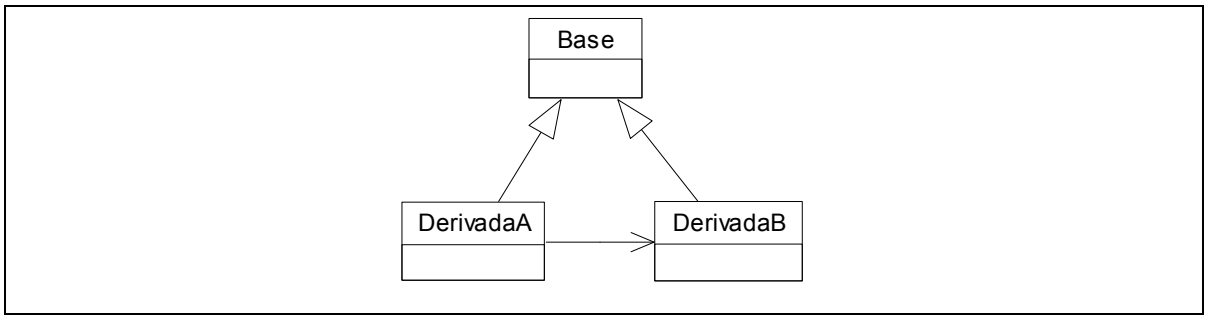



*Figura 6. 20: Estrutura geral do padrão Derivadas Homólogas*

O padrão mais simples possível obedece à estrutura da Figura 6. 21.

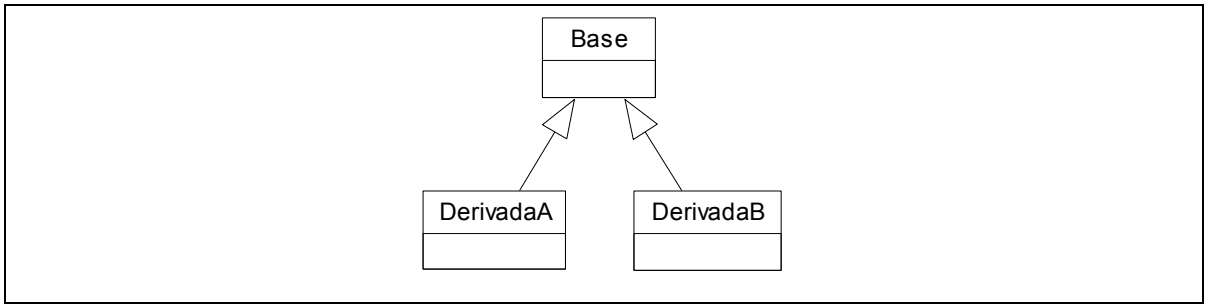



*Figura 6. 21: Estrutura simples do padrão Derivadas Homólogas*

Na Figura 6. 22 temos a combinação do padrão Derivadas Homólogas com os padrões Referência Múltipla e Árvore.

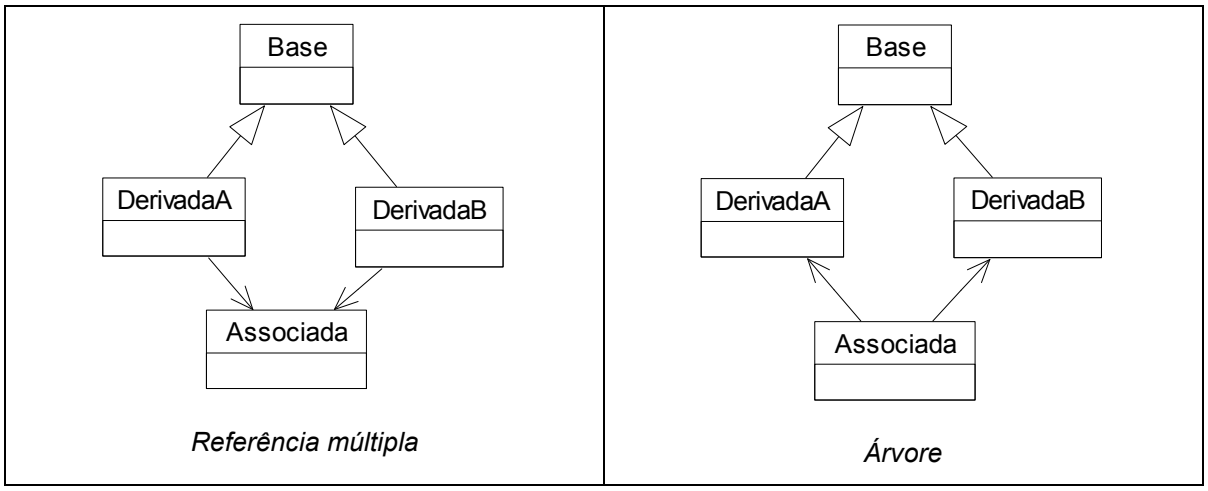



*Figura 6. 22: Combinação Derivadas Homólogas com Referência Múltipla e Árvore*

Entretanto, como foi comentado, essa combinação não será tratada como um novo padrão estrutural. Para fins de análise de padrões estruturais, é importante revelar primariamente os padrões mais simples.

**Participantes**

- *Base*: classe base que possui duas ou mais classes derivadas.
- *Derivadas*: classes derivadas que podem ou não estarem associadas entre si.

**Exemplo**

O diagrama de classes da Figura 6. 23 mostra o mapeamento UML para o esquema XML usado pela OASIS Business Process Execution Language (BPEL).

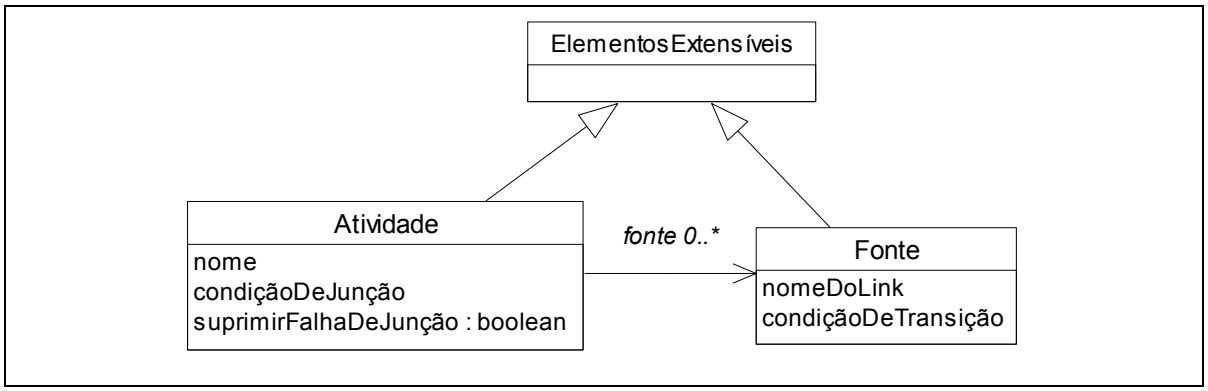



*Figura 6. 23: Padrão encontrado no modelo da OASIS Business Process Execution Language*

Nesse modelo, ambas as classes `Atividade` e `Fonte` são derivadas da mesma classe base, assim temos representações de conceitos semelhantes nas classes herdeiras. Além disso, existe um relacionamento entre esses conceitos parecidos, feito por meio da associação `fonte`.

### 6.4.3 Herança Multinível

**Intenção**

Criar uma estrutura de árvore de herança, onde é necessário mais de um nível de especialização.

**Descrição**

Em estruturas que envolvem árvores de herança esse padrão pode ser comum. Sua idéia é aplicada sempre que o projetista quer especializar conceitos relacionados, e a especialização exige mais de um nível de relacionamento de herança.

O padrão pode estar inserido no contexto de uma estrutura que envolve outros padrões relacionados à herança. Assim podemos ter, por exemplo, o padrão Derivadas Homólogas e o padrão Herança Multinível, num mesmo modelo.

**Estrutura**

A estrutura geral desse padrão é mostrada na Figura 6. 24.

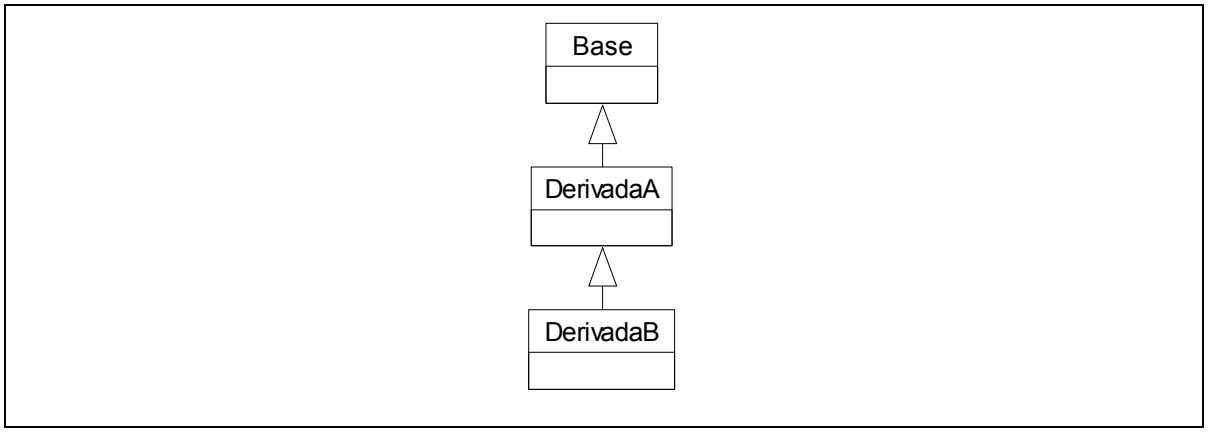



*Figura 6. 24: Estrutura do padrão Herança Multinível*

**Participantes**

- *Base*: classe raiz da árvore de herança
- *Derivadas*: classes intermediárias, relacionadas a cada nível de especialização da árvore

**Exemplo**

O diagrama de classes da Figura 6. 25 mostra o mapeamento UML para o esquema XML usado pelo W3C XML Schema for Schemas.

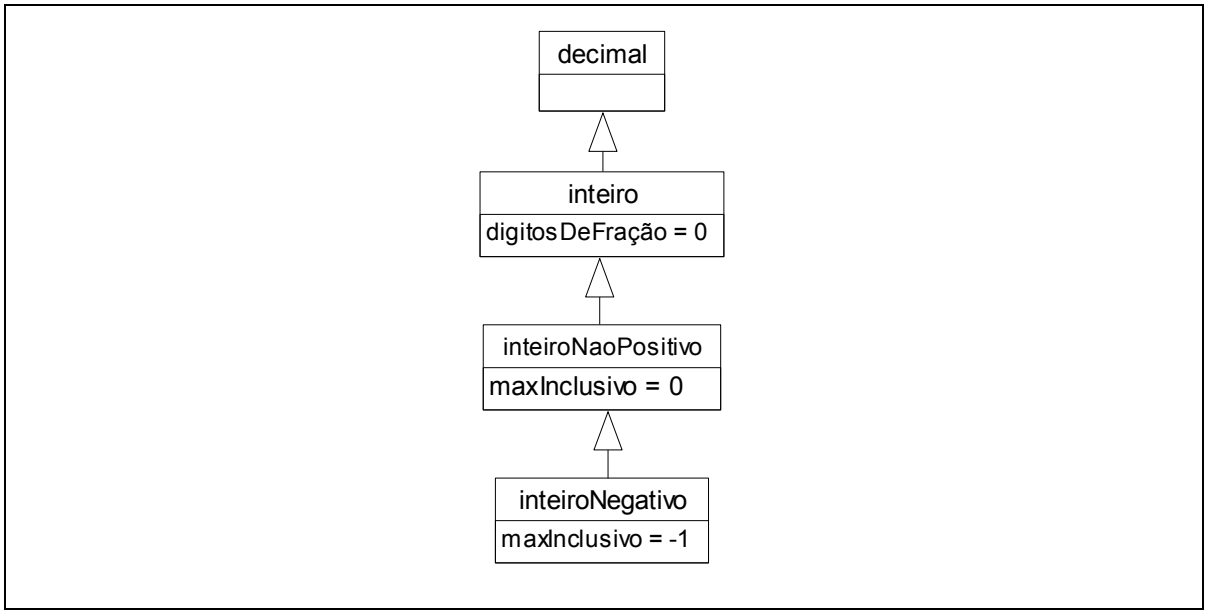



*Figura 6. 25: Padrão encontrado no modelo do W3C XML Schema for Schemas*

A hierarquia de especializações mostra como um número `decimal` pode ser especializado em diversos níveis. Passando pelas classes `inteiro` e `inteiroNaoPositivo`, o último nível de especialização representa um `inteiroNegativo`.

### 6.4.4 Associação Derivada-Base

**Intenção**

Relacionar um conceito, representado por uma especialização, à sua própria classe base.

**Descrição**

Uma classe derivada pode ter uma relação de herança e, ao mesmo tempo, associação com uma classe base. Esse padrão parece não estar muito presente nos esquemas analisados, entretanto ele apresenta um caso específico que deve ser estudado.

Como o padrão possibilita que a classe derivada seja associada à sua classe base, apresenta um caso de associação não tratado pelo padrão Herança-Associação, onde a classe em que ocorre a relação de associação não faz parte da hierarquia de herança.

A organização das classes é interessante para aplicações sobre padrões. Isso advém do fato de que a classe derivada representa um elemento que pode conter elementos muito parecidos consigo mesmo (pois seguem a forma da classe base).

**Estrutura**

A estrutura do padrão é representada na Figura 6. 26.

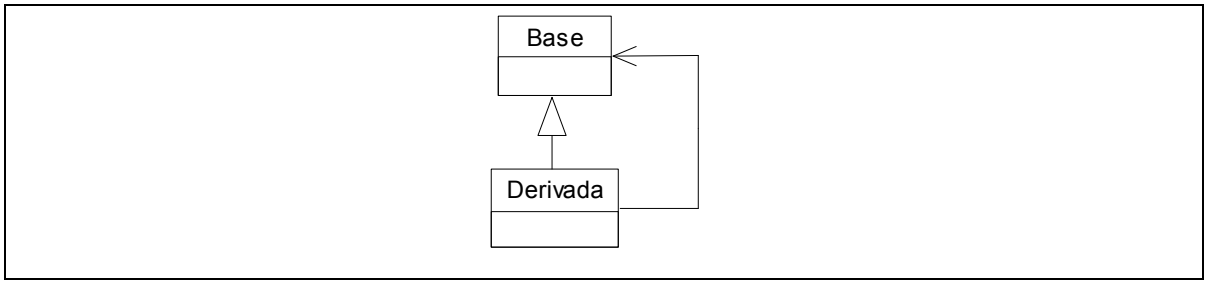



*Figura 6. 26: Estrutura do padrão Associação Devivada-Base*

**Participantes**

- *Base*: classe base que é referenciada pela classe derivada
- *Derivada*: classe derivada que possui uma associação referenciando a classe base

**Exemplo**

O diagrama de classes da Figura 6. 27 mostra o mapeamento UML para o esquema XML usado pela *Open Applications Group* (OAGIS).

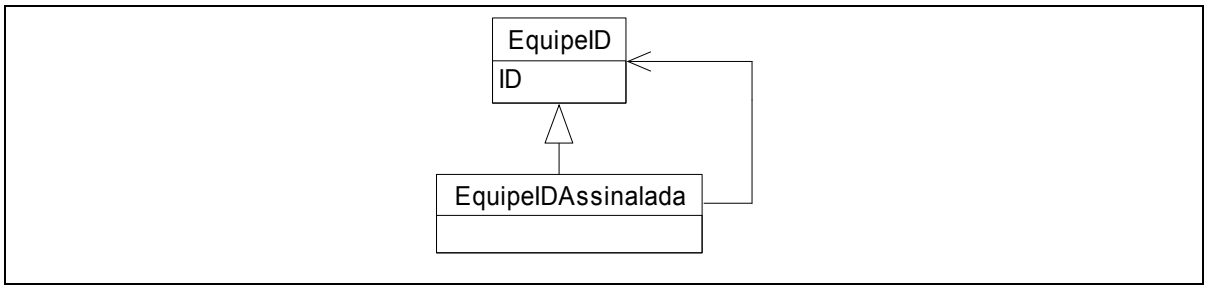



*Figura 6. 27: Padrão encontrado no modelo da Open Applications Group (OAGIS)*

Uma EquipeID é base para a criação da classe EquipeIDAssinalada. Por sua vez, a classe derivada possui associação com a classe base, o que permite que uma equipe assinalada possa referenciar uma equipe mais geral.

## 6.5 Considerações Finais

Nesse capítulo foram apresentados padrões estruturais XML baseados em modelos UML, que foram definidos a partir da análise de vocabulários XML encontrados em modelos reais. Foi construído um catálogo desses padrões, divididos em padrões de associação e padrões de herança. Buscou-se capturar a semântica revelada pela estrutura de cada padrão, pois o significado contido em cada padrão poderá ser útil para aplicações que os utilizem. A Tabela 6. 3 apresenta um resumo dos padrões.

A informação semântica revelada nas estruturas definidas pelos padrões possibilita a construção de formas de análise do vocabulário XML. Uma aplicação possível poderia ser a verificação da qualidade do vocabulário, a partir de boas (ou más) técnicas de modelagem. Outra aplicação promissora é explorar o uso da análise dos padrões no auxílio à atividade de testes. O Capítulo 7 utiliza os padrões aqui propostos para definir técnicas de geração de dados de teste aplicados à comunicação de componentes que trocam mensagens XML. A elaboração de dados de teste para aplicações que usam um vocabulário XML pode se beneficiar do uso dos padrões, a partir da descoberta de informações relevantes no modelo UML. Dessa maneira podem-se gerar dados de teste significativos, cujo objetivo é revelar defeitos específicos.

*Tabela 6. 3: Padrões Estruturais XML*

| **Padrões de Associação** | |
|---|---|
| **Árvore** | Modela uma hierarquia de classes como uma árvore |
| **Associações Homólogas** | Expressa a associação entre dois conceitos em diferentes contextos |
| **Associação Recursiva** | Modela o relacionamento entre elementos definido de uma maneira recursiva |
| **Associação Bi-direcional** | Modela o relacionamento entre dois elementos, onde o primeiro referencia o segundo e vice-versa |
| **Referência Múltipla** | Modela, como uma classe singular, um elemento que pode ser referenciado por múltiplas classes |
| **Atalho** | Referencia um conceito utilizando diferentes níveis de associaçãoes |
| **Ciclo** | Cria um ciclo de associações |
| **Padrões de Herança** | |
| **Associação Derivada-Base** | Cria um relacionamento entre um conceito e sua classe base |
| **Derivadas Homólogas** | Estabelece um grupo de classes derivadas de uma mesma classe base, o qual pode apresentar relacionamentos entre seus elementos |
| **Herança Multi-Nível** | Cria uma hierarquia de classes com mais de um nível |
| **Herança-Associação** | Estabelece uma associação específica entre classes base (ou derivadas) com outras classes |

*Qualquer problema de programação pode ser resolvido adicionando-se um nível de indireção. (veja também: "Qualquer problema de performance..." por M. Haertel)*
***Anônimo***

# *7 Perturbação de Dados Dirigida por Padrões*

Foram analisados no Capítulo 4 os conceitos relativos à técnica de perturbação de dados. Foram mostradas as características das perturbações definidas por Offutt e Wuzhi [OW04]. Uma limitação apresentada por essas perturbações é que elas não consideram uma análise semântica do vocabulário utilizado na comunicação dos componentes.

Nesse capítulo são propostas perturbações de dados que se baseiam nos padrões estruturais definidos anteriormente. Com a perturbação de dados dirigida por padrões é possível gerar dados de teste que consideram a semântica do vocabulário utilizado na comunicação de componentes. Essa proposta representa um avanço em relação às técnicas de perturbação propostas por Offutt e Wuzhi, que analisam somente informações sintáticas do modelo. A seguir é apresentada a proposta de perturbação de dados dirigida por padrões, buscando-se comparar suas vantagens e complementações em relação a outras formas de perturbação.

## 7.1 Princípios da Perturbação Dirigida por Padrões

Para demonstrar o uso dos padrões estruturais na atividade de teste, foram desenvolvidos algoritmos de perturbação baseados nos padrões definidos. Como foi visto, os padrões estruturais foram propostos com a intenção de revelar a semântica envolvida na modelagem de vocabulários XML. Com a ajuda dos padrões, pode-se analisar parte do sentido que as mensagens XML apresentam durante a comunicação dos componentes.

O uso da perturbação dirigida por padrões permite a geração de dados de teste, onde o significado do vocabulário XML é considerado. Tem-se assim uma grande chance de se gerar dados de teste significativos, que podem revelar defeitos complementares aos revelados com os operadores sintáticos.

A perturbação dirigida por padrões é aplicada com base na existência de um vocabulário XML. A técnica proposta é adequada para aplicações que utilizam vocabulários XML modelados utilizando-se a linguagem UML, como foi visto anteriormente. Dessa maneira pode-se explorar de forma significativa o poder de expressão da modelagem UML/XML.

Para se gerar uma mensagem perturbada é preciso analisar o vocabulário XML (definido em um esquema). Além disso é necessário ter-se instâncias de elementos definidos em termos do vocabulário, que serão utilizadas para a substituição ou adição de elementos nas mensagens.

Dada uma mensagem XML, o processo de perturbação é realizado da seguinte forma. Busca-se na mensagem algum elemento XML que seja de interesse para se aplicar o operador de perturbação, definido conforme um princípio que tem por base um padrão. Caso se tenha detectado algum elemento de interesse, realiza-se uma troca desse elemento por outro elemento específico; também é possível apenas adicionar elementos sem substituição. Dessa maneira podem-se gerar mensagens perturbadas, que serão utilizadas para os testes. A seguir são descritos os princípios em que se baseiam as perturbações propostas. Os algoritmos para encontrar elementos para a substituição que a perturbação realiza encontram-se no Apêndice B. Esses algoritmos foram implementados em Java (cf. Capítulo 8); seu pseudo-código, presente no apêndice, mostra em alto nível os principais passos da perturbação. Ao final de cada descrição são mostrados exemplos da aplicação dos princípios de perturbação.

## 7.2 Perturbação Associações Homólogas

O padrão Associações Homólogas reconhece duas ou mais classes distintas que se relacionam por meio de duas ou mais associações, as quais revelam o significado específico dos relacionamentos. A Figura 7. 1 mostra essa estrutura.

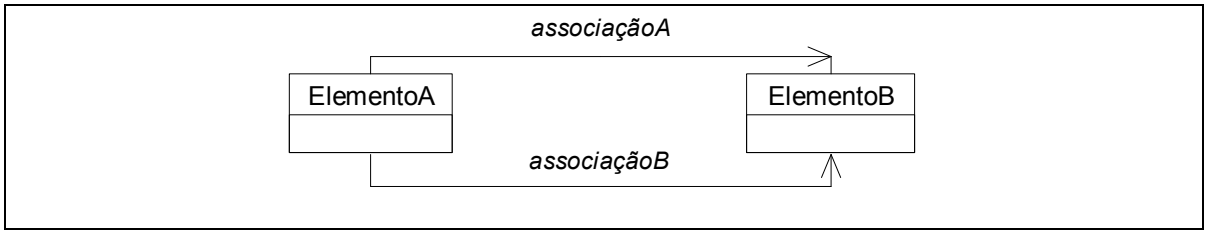



*Figura 7. 1: Estrutura do padrão Associações Homólogas*

A perturbação nos dados da mensagem XML é baseada na troca dos elementos (atributos de uma classe) relacionados a uma determinada associação pelos dados relativos à outra

associação homóloga arbitrária. A partir de uma mensagem XML inicial, é criada uma mensagem XML perturbada, onde os parâmetros que no mapeamento UML se relacionam aos atributos da classe que recebe associações homólogas são modificados. Os novos valores para esses atributos são encontrados em instâncias que são definidas em termos das outras associações homólogas dessa estrutura. Dessa maneira, a semântica da mensagem é alterada para uma semântica diferente, mas possivelmente muito próxima e coerente com relação ao significado original.

### 7.2.1 Exemplo[7]

Será considerado o exemplo ilustrado na Figura 7. 2: Fragmento do vocabulário da OASIS Universal Business Language, que foi extraído do vocabulário da OASIS Universal Business Language (UBL).

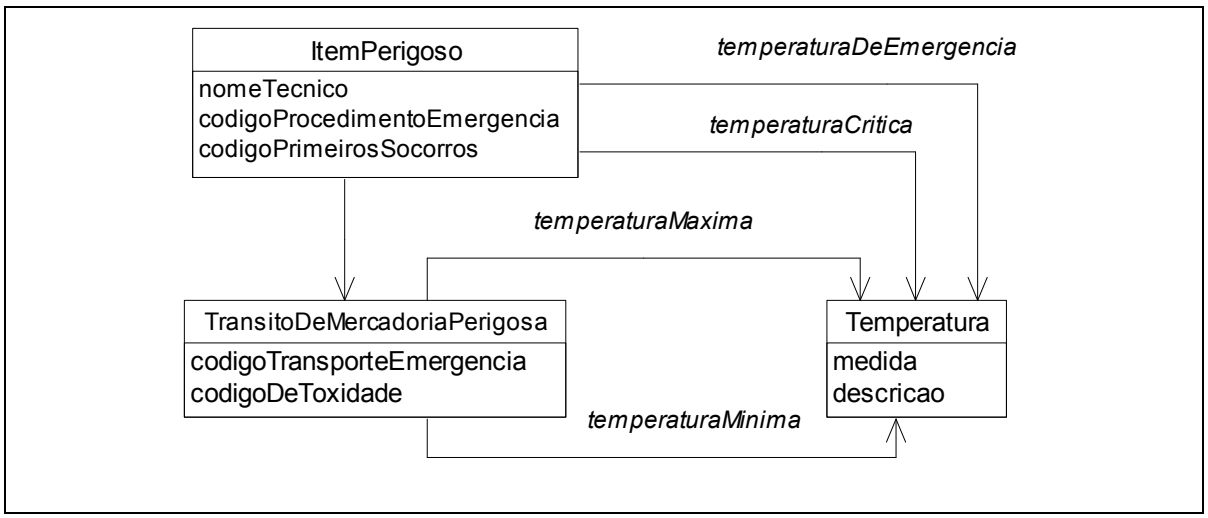



*Figura 7. 2: Fragmento do vocabulário da OASIS Universal Business Language*

O vocabulário modela o transporte de mercadorias perigosas, onde é importante controlar a temperatura das mercadorias. Podemos identificar dois conjuntos de associações homólogas; o primeiro contém as associações `temperaturaMaxima` e `temperaturaMinima`, o segundo

[7] Os exemplos buscam mostrar como os princípios de perturbação dirigida por padrões podem gerar dados de teste úteis para a descoberta de defeitos. Para isso os exemplos são baseados nos vocabulários reais apresentados no Capítulo 5. Como foi comentado anteriormente, nem sempre os dados de teste gerados são ótimos para a detecção de defeitos; os exemplos a seguir mostram casos em que os dados gerados tem grande probabilidade de encontrar defeitos na implementação.

contém `temperaturaCritica` e `temperaturaDeEmergencia` (note-se que o modelo também apresenta o padrão Referência Múltipla, onde diversas associações referenciam a classe `Temperatura`; esse padrão será tratado na seção a seguir).

O algoritmo de perturbação aplicado a esse exemplo gera, entre outras, uma mensagem de teste que envolve as associações `temperaturaMaxima` e `temperaturaMinima`. Uma mensagem relativa ao transporte de um produto perigoso poderia apresentar o seguinte trecho, onde se tem definidas as temperaturas (atributos de codificação e serialização foram omitidos, para simplificar a mensagem):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<soapenv:Envelope ...>
  <soapenv:Body>
    <ns1:processaItemPerigoso ...>
      <argumento href="#id0" />
    </ns1:processaItemPerigoso>

    <multiRef id="id0" ...>
      <nomeTecnico xsi:type="soapenc:string">Nitrato de Chumbo II</nomeTecnico>
      <temperaturaMaxima href="#id1" />
      <temperaturaMinima href="#id2" />
    </multiRef>

    <multiRef id="id1" ...>
      <medida xsi:type="soapenc:string">27</medida>
    </multiRef>

    <multiRef id="id2"...">
      <medida xsi:type="soapenc:string">10</medida>
    </multiRef>
   </soapenv:Body>
</soapenv:Envelope>
```

Suponha que o elemento escolhido para ser perturbado seja a `medida` referente a `temperaturaMaxima`, que tem valor 27. De acordo com o princípio de perturbação, o valor do elemento `medida` escolhido será substituído pelo conteúdo de um elemento `medida` relacionado a um elemento `temperaturaMinima`, encontrado em alguma instância da biblioteca de documentos XML. Considere que o seguinte trecho de instância XML pertence à biblioteca:

```
<ItemPerigoso>
  <nomeTecnico>Tetraóxido de Rutênio</nomeTecnico>
  ...
  <TransitoDeMercadoriaPerigosa>
    <codigoTransporteEmergencia>33</codigoTransporteEmergencia>
    ...
    <temperaturaMaxima>
      <medida>45</medida>
      ...
    </temperaturaMaxima>
    <temperaturaMinima>
      <medida>-7</medida>
      ...
    </temperaturaMinima>
  </TransitoDeMercadoriaPerigosa>
</ItemPerigoso>
```

Ao aplicar a perturbação usando-se a instância anterior, o valor da `medida` "27" referente a `temperaturaMaxima` da mensagem seria substituído pelo valor da `medida` "-7" referente a `temperaturaMinima`. A mensagem resultante apresentaria os seguintes valores:

```
...
<multiRef id="id0" ...>
  <nomeTecnico xsi:type="soapenc:string">Nitrato de Chumbo II</nomeTecnico>
  <temperaturaMaxima href="#id1" />
  <temperaturaMinima href="#id2" />
</multiRef>

<multiRef id="id1" ...>
  <medida xsi:type="soapenc:string">-7</medida>
</multiRef>

<multiRef id="id2"...">
  <medida xsi:type="soapenc:string">10</medida>
</multiRef>
...
```

Observa-se que na mensagem obtida com a perturbação dos dados a temperatura máxima tem um valor de medida menor que o valor da temperatura mínima. Essa situação pode, por exemplo, simular um engano cometido pelo usuário, durante o preenchimento dos dados do produto.

Essa instância de perturbação dos dados da mensagem poderia ser útil para testar e assegurar a confiabilidade do tratamento de produtos perigosos de um sistema. A *Univeral Business Language* (UBL) define somente o vocabulário da linguagem de comercialização de produtos. Uma implementação específica irá definir, em seu código, o tratamento dos dados. A confiabilidade desse tratamento poderia ser avaliada com a instância resultante da perturbação, pois uma implementação segura do sistema pode se preocupar em tratar temperaturas inconsistentes, como é o caso onde a temperatura mínima de segurança apresenta-se maior que a temperatura máxima.

## 7.3 Perturbação Referência Múltipla

Como foi visto, o padrão Referência Múltipla define uma classe singular a ser referenciada por um número arbitrário de outras classes, por meio de associações. A Figura 7. 3 mostra essa estrutura.

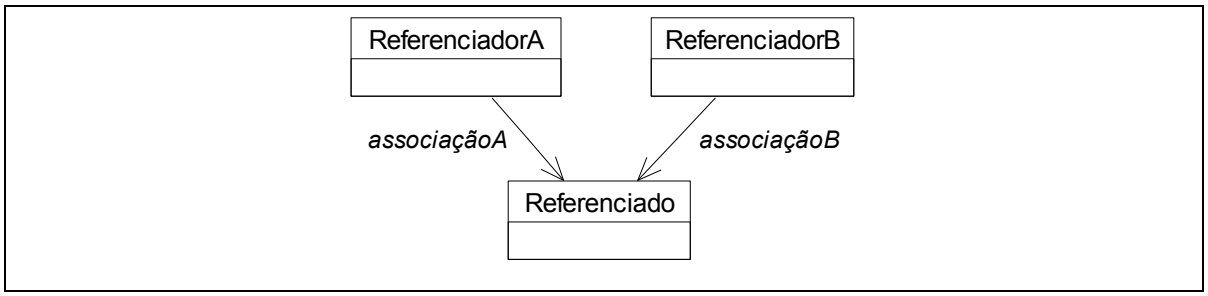



*Figura 7. 3: Estrutura do padrão Referência Múltipla*

Para se realizar a perturbação nos dados da mensagem, é definida uma modificação em elementos XML associados a instâncias de classes singulares referenciadas. Dada uma instância de classe singular relativa a uma associação, seus atributos (elementos XML) são alterados; a origem dos novos valores é associada à outra instância de classe referente a uma associação de nome desigual. O resultado é uma instância modificada que mantém uma semântica coerente com a definição do vocabulário.

### 7.3.1 Exemplo

Será analisado o exemplo ilustrado na Figura 7. 4, que foi extraído do vocabulário da OASIS Universal Business Language (UBL).

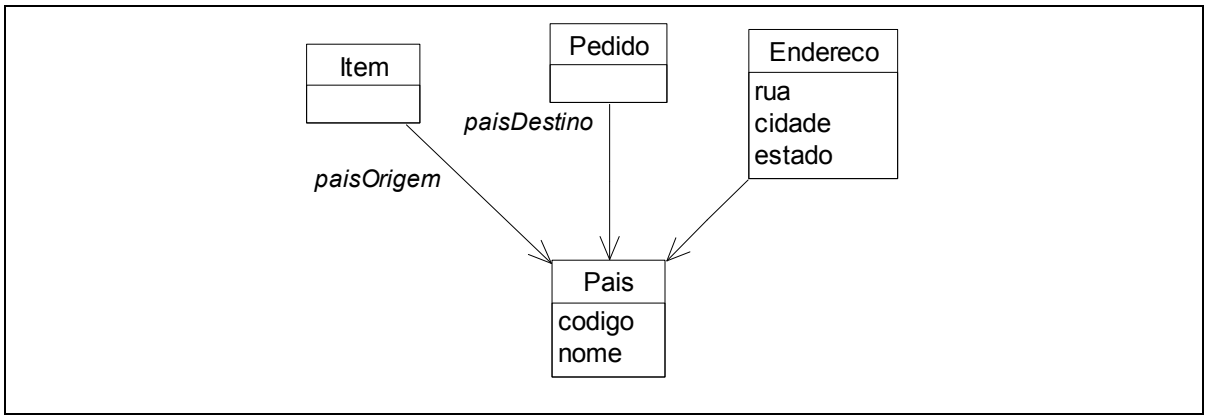



*Figura 7. 4: Fragmento do vocabulário da OASIS Universal Business Language*

A classe singular `Pais` é usada por três outros conceitos. Considere, por exemplo, a classe `Pais` sendo utilizada numa associação com a classe `Endereco`, para modelar o endereço de um cliente de uma loja virtual.

Uma mensagem que se utiliza desse vocabulário poderia estar relacionada ao fornecimento do endereço de entrega de um produto para o cliente. Considere que os elementos presentes na seguinte mensagem SOAP são objetos de comunicação de alguma mensagem:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope ... >
  <SOAP-ENV:Body>
    <ns1:informaEndereco xmlns:ns1=... >
      <rua>Ceilão</rua>
      <estado>Paraná</estado>
      <pais>
        <codigo>19</codigo>
        <nome>Brasil</nome>
      </pais>
    </ns1:informaEndereco>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Pode-se aplicar a perturbação nos dados da mensagem, nos elementos que são mapeados para atributos da classe `Pais`, no mapeamento UML. Por exemplo, o elemento `nome` (nome do país) poderá ser substituído por outro valor durante a perturbação. O valor escolhido poderá ser o país de origem de um `Item`, por exemplo, se o seguinte fragmento fizer parte da biblioteca de documentos XML:

```
<Item>
  ...
  <paisOrigem>
    <codigo>33</codigo>
    <nome>Taiwan</nome>
  </paisOrigem>
</Item>
```

Nesse caso, temos a geração de uma mensagem onde o valor do país original de entrega “Brasil” será substituído pelo valor correspondente a “Taiwan”. Essa perturbação pode simular uma tentativa de fornecer dados incoerentes, feita por um usuário mal intencionado ou simplesmente desatento.

Um sistema de comércio eletrônico, que seja implementado utilizando o vocabulário UBL, poderia apresentar um defeito caso a validação dos dados relativos ao endereço de entrega não fosse realizada de forma correta. Desse modo a perturbação nos dados dessa mensagem fictícia gera dados de teste que podem revelar defeitos na implementação de um sistema que utiliza o vocabulário em questão.

## 7.4 Perturbação Associação Recursiva

A estrutura do padrão Associação Recursiva identifica um conceito que mantém associações com conceitos de seu mesmo tipo. O padrão permite uma organização hierárquica entre conceitos semelhantes. A Figura 7. 5 mostra essa estrutura.

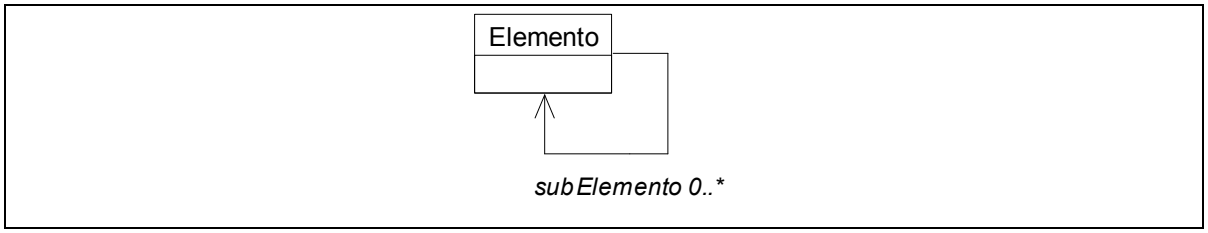



*Figura 7. 5: Estrutura do padrão Associação Recursiva*

Uma instância que atenda ao padrão possui a forma de uma árvore hierárquica para um conjunto de conceitos definidos nos mesmos termos. A perturbação nos dados define uma alteração nessa hierarquia. Para um dado elemento na árvore, serão considerados dois tipos de substituição: 1) substituição por um elemento anterior na hierarquia; 2) substituição por um elemento posterior na hierarquia. Esses dois casos podem apresentar resultados diferentes, dependendo do significado que a hierarquia apresenta.

### 7.4.1 Exemplo

Será estudado o exemplo mostrado na Figura 7. 6, que tem origem no vocabulário da eBay XML API.

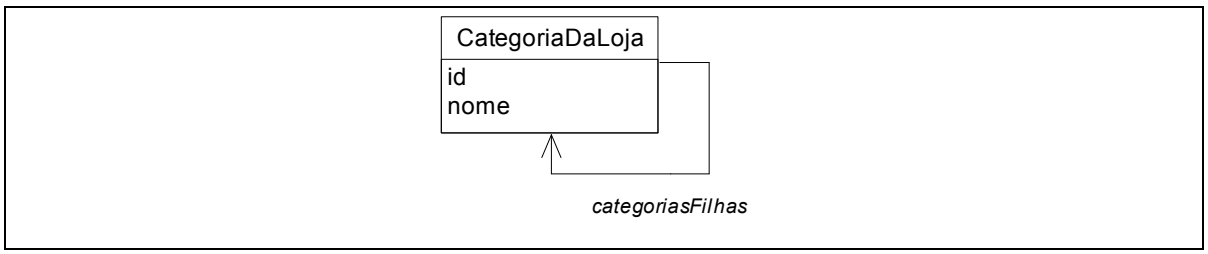



*Figura 7. 6: Fragmento do vocabulário da eBay XML API*

A loja virtual está dividida em categorias, onde cada categoria pode apresentar categorias filhas. Um exemplo de hierarquia de categorias seria “computadores”, que inclui “componentes”, que inclui “monitores”. Uma instância na forma do vocabulário poderia ser:

```
<CategoriaDaLoja>
  <nome>computadores</nome>
  <categoriasFilhas>
    <nome>componentes</nome>
    <categoriasFilhas>
      <nome>monitores</nome>
    </categoriasFilhas>
  </categoriasFilhas>
</CategoriaDaLoja>
```

Suponha que uma mensagem arbitrária possui um processamento sobre o elemento `nome` para alguma categoria, como nesse fragmento:

```
<SOAP-ENV:Envelope ...>
  <SOAP-ENV:Body>
    <ns1:processaCategoria xmlns:ns1="http://soapinterop.org/">
      <nome xsi:type="xsd:string">processadores</nome>
      ...
    </ns1: processaCategoria >
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Considere que essa categoria se refere a um processador qualquer, onde a categoria anterior é “computadores”. Uma aplicação da perturbação nos dados da mensagem poderia substituir o valor “processadores” por “computadores”; ou mesmo por um grupo de processadores específicos, por exemplo, “processadores de 32 bits”.

Para o caso em que é criada uma mensagem onde o valor “processadores” foi substituído pelo valor “computadores”, observamos que a relação de inclusão entre os conjuntos permanece com sentido para uma árvore XML com raiz em “computadores” (pois um processador faz parte da categoria dos computadores). Já uma substituição por um item posterior na hierarquia poderia criar uma mensagem semanticamente incorreta do ponto de vista da inclusão dos conjuntos, por exemplo um estado onde “processadores de 64 bits” inclui “processadores de 32 bits”.

O comportamento do sistema, diante dessas duas possibilidades, é um bom alvo a ser testado pela mensagem XML resultante da perturbação. Um sistema específico poderia ter o requisito de ser robusto para tratar uma organização hierárquica incorreta; ou mesmo para prever uma variação na hierarquia, onde se mantém correta a relação de inclusão dos subconjuntos de categorias filhas.

## 7.5 Perturbação Derivadas Homólogas

O padrão Derivadas Homólogas mostra um conjunto de classes derivadas de uma mesma classe base, as quais podem apresentar associações entre si. A Figura 7. 7 ilustra o caso simples com duas classes derivadas homólogas.

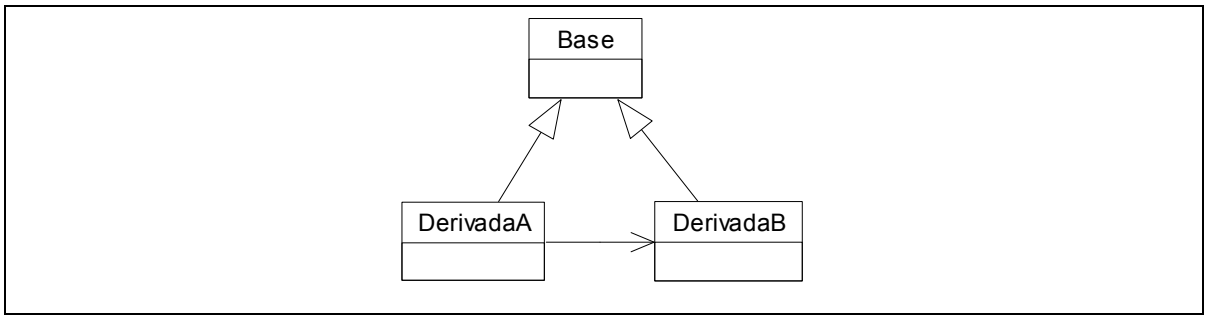



*Figura 7. 7: Estrutura do padrão Derivadas Homólogas*

As instâncias de documentos XML que apresentam o padrão podem ter seus dados perturbados por meio de uma operação de “cruzamento” dos dados. As classes derivadas apresentam um estado específico (atributos da classe derivada) e um estado comum (atributos da classe base). A operação de perturbação dos dados é definida com uma substituição do valor do estado comum de uma classe derivada pelo estado comum de uma classe derivada homóloga.

Esse “cruzamento” de dados permite que sejam criadas instâncias significativas, representando conceitos que têm grande probabilidade de serem dados de teste úteis. Esse fato

está relacionado à semântica envolvida na definição de uma estrutura de herança, onde o estado comum das classes merece atenção nos dados de teste.

### 7.5.1 Exemplo

Será analisado como exemplo um trecho do vocabulário do W3C XML Schema for Schemas, ilustrado na Figura 7. 8.

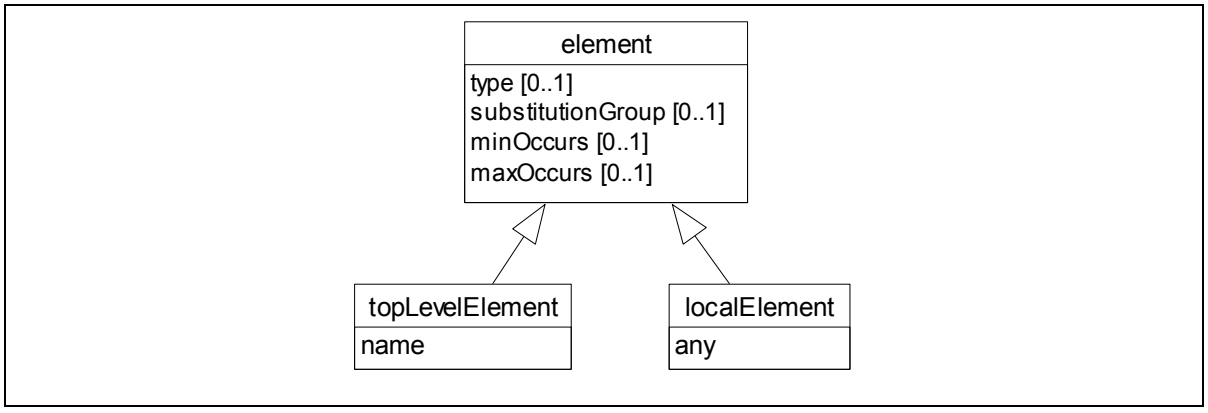



*Figura 7. 8: Fragmento do vocabulário XML Schema for Schemas*

Existem duas classes derivadas de `element`, que possuem aspectos bastante comuns com a classe base. Nota-se que `localElement` permite um tipo `any`; enquanto `topLevelElement` tem o atributo `name` como não-opcional.

Esse vocabulário poderia ser utilizado para uma aplicação que manipula documentos XML Schema. Para um exemplo de perturbação nos dados, considere que o fragmento XML a seguir é utilizado na comunicação entre dois componentes dessa aplicação.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<soapenv:Envelope ...>
  <soapenv:Body>
    <ns1:processaEsquema ...>
      <topLevelElement href="#id0" />
    </ns1:processaEsquema>

    <multiRef id="id0" ...>
      <type xsi:type="soapenc:string">Classe1</type>
      <minOccurs xsi:type="soapenc:string">0</minOccurs>
      <maxOccurs xsi:type="soapenc:string">1</maxOccurs>
      <name xsi:type="soapenc:string">classe1</name>
    </multiRef>
   </soapenv:Body>
</soapenv:Envelope>
```

Esse trecho poderia se referir à definição de um elemento qualquer, onde se percebe que sua multiplicidade, definida por `minOccurs` e `maxOccurs`, torna-o optativo. A perturbação dos dados poderia ser feita utilizando-se o seguinte fragmento de instância:

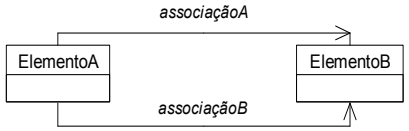
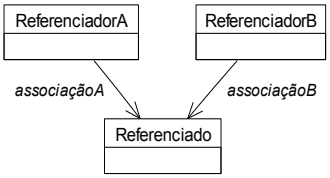
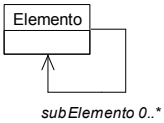
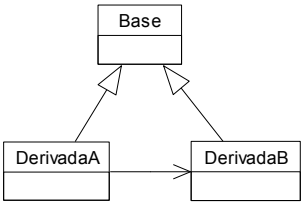
```
<localElement>
  ...
  <minOccurs>1</minOccurs>
  <maxOccurs>unbounded</maxOccurs>
  ...
</localElement>
```

Como resultado, obtém-se uma nova instância de mensagem onde possivelmente elementos como `type` ou `name` são modificados (dependendo da seleção dos elementos a serem perturbados, definida pelo usuário). Observe que a multiplicidade poderia ser alterada para `minOccurs` tendo valor “1” e `maxOccurs` tendo valor “unbounded”. Todas essas mudanças podem levar à concepção de uma mensagem XML com características relevantes para o teste da aplicação. A multiplicidade, por exemplo, poderia ser um fator a ser considerado, dependendo dos objetivos da aplicação, onde as restrições sobre os elementos podem ter como fator crítico o número máximo ou mínimo de itens permitidos.

## 7.6 Considerações Finais

Nesse capítulo foi apresentado o conceito de perturbação de dados dirigida por padrões. Foram definidos operadores de perturbação baseados nos padrões estruturais propostos no Capítulo 6. A Tabela 7. 1 apresenta um resumo de cada operador de perturbação proposto.

*Tabela 7. 1: Operadores de Perturbação*

| Op. | Descrição | Estrutura do Padrão UML |
|---|---|---|
| Associações Homólogas | A perturbação nos dados da mensagem XML é baseada na troca dos valores dos elementos (atributos de uma classe) relacionados a uma determinada associação pelos dados relativos à outra associação homóloga arbitrária. |  |
| Referência Múltipla | Dada uma instância de classe referenciada (singular) relativa a uma associação, seus atributos (elementos XML) são substituídos pelos atributos de uma instância de classe referente a uma outra associação que referencia a classe singular. O resultado é uma instância modificada que mantém uma semântica coerente com a definição do vocabulário. |  |
| Associação Recursiva | A perturbação é baseada numa alteração na hierarquia da árvore XML. Para um dado elemento na árvore, pode-se substituir seu valor pelo valor de um elemento anterior ou posterior na hierarquia. |  |
| Derivadas Homólogas | As classes derivadas apresentam um estado específico (atributos da classe derivada) e um estado comum (atributos da classe base). A operação de perturbação dos dados é definida com uma substituição do valor do estado comum de uma classe derivada pelo estado comum de uma classe derivada homóloga. |  |

Para explorar a abordagem proposta, foi desenvolvida uma ferramenta de testes baseada nas perturbações definidas. O próximo capítulo mostrará como os conceitos propostos foram utilizados para a implementação dessa ferramenta. Em seguida, no Capítulo 9, será abordada a utilização da ferramenta e dos operadores propostos em um sistema real, com o qual foi realizado um estudo de caso.

*Programas devem ser escritos para as pessoas lerem, e casualmente para máquinas executarem.*
***H. Abelson e G. Sussman***

# *8 Uma Ferramenta para Teste Baseado em Perturbação de Dados Dirigida por Padrões*

Para possibilitar uma melhor análise dos conceitos de perturbação de dados propostos, com sua aplicação prática, foi desenvolvida uma ferramenta para teste baseado em perturbação de dados dirigida por padrões, que foi chamada de PDDP (sigla de Perturbação de Dados Dirigida por Padrões). A ferramenta não tem o intuito de ser usada em um ambiente de produção, mas sim num ambiente de pesquisa, para a validação dos aspectos teóricos.

Assim a ferramenta foi desenvolvida com a finalidade de explorar os conceitos propostos, a fim de avaliar sua aplicabilidade. Esse capítulo apresenta os detalhes de como a PDDP foi implementada e como os conceitos teóricos foram colocados em prática. A arquitetura da ferramenta é apresentada de maneira a mostrar os detalhes de implementação relevantes.

## 8.1 Considerações iniciais

A ferramenta desenvolvida é voltada para o teste de *Web Services*, onde se utilizam os conceitos de perturbação para gerar dados de teste em XML, de acordo com o protocolo utilizado na comunicação dos componentes, o *Simple Object Access Protocol* (SOAP) [W3C03]. Entretanto os conceitos poderiam ser aplicados de maneira semelhante para outros protocolos, dado que o formato das mensagens é um XML arbitrário que é encapsulado no protocolo SOAP.

Não se espera prender os princípios teóricos a uma tecnologia específica. Outros tipos de comunicação XML arbitrários poderiam ser tratados, não necessariamente em uma implementação com *Web Services*. Essa tecnologia foi escolhida devido à sua importância atual, popularidade e conveniência para implementação.

A seguir é mostrada a visão da ferramenta, com a definição do domínio de aplicação, requisitos do sistema, funcionalidades disponíveis e componentes da arquitetura do sistema.

## 8.2 Visão da Ferramenta

Os artefatos fundamentais para a PDDP são: a mensagem SOAP da qual são derivadas as perturbações; a definição do vocabulário utilizado na comunicação dos componentes; uma biblioteca de instâncias do vocabulário, utilizada para derivar as mensagens perturbadas.

A mensagem SOAP inicial é utilizada como modelo para criar as mensagens perturbadas, onde pelo menos um elemento da sub-árvore XML que é encapsulada no elemento *Body* do protocolo SOAP é modificado.

A definição do vocabulário é feita utilizando-se as técnicas de modelagem UML estudadas anteriormente. Como a ferramenta de apoio utilizada foi a hyperModel, dirigida ao método de mapeamento de Carlson [Car01], o sistema é compatível com a definição de vocabulário dessa ferramenta. Para isso é conveniente usar o arquivo de definição do XML Schema gerado pela hyperModel, pois esse esquema segue um padrão de definição dos elementos XML bem construído, segundo o mapeamento UML/XML. A *Figura 8. 1* mostra a modelagem de um diagrama de classes no hyperModel (esquerda), assim como o respectivo XML Schema (direita).

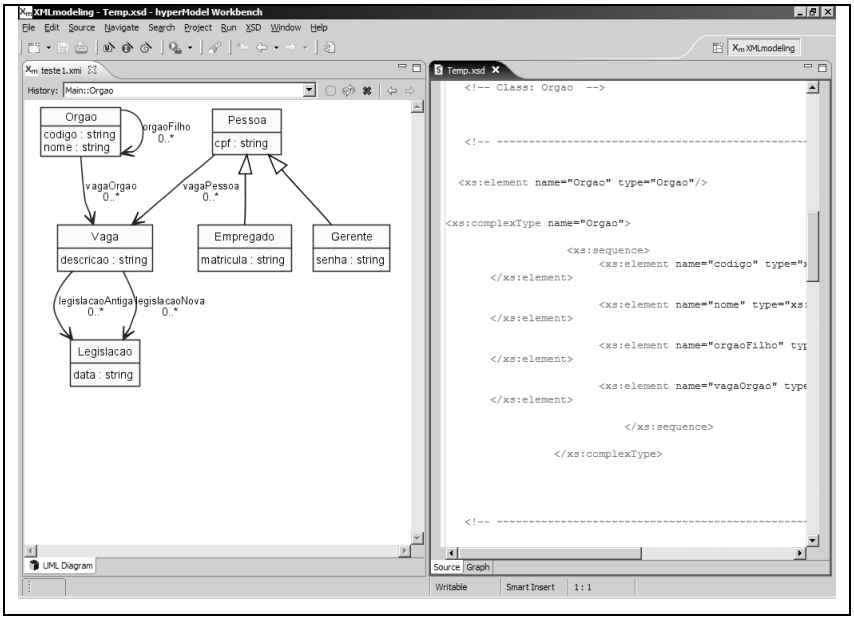



*Figura 8. 1: Modelagem de diagrama de classes na hyperModel*

A biblioteca de instâncias que obedece ao vocabulário definido é a fonte de elementos utilizados para a geração de mensagens perturbadas. É importante se ter uma biblioteca que expresse de forma completa a semântica do vocabulário, com instâncias de elementos XML refletindo os relacionamentos definidos.

A Figura 8. 2 apresenta o relacionamento entre os artefatos e as ferramentas. As setas indicam os principais relacionamentos entre os elementos. O vocabulário é gerado na hyperModel e fornecido à PDDP. A biblioteca de instâncias deve obedecer ao vocabulário (no experimento, descrito no próximo capítulo, a biblioteca foi gerada com uma seleção aleatória de elementos fornecidos pelo testador; esses elementos poderiam ser obtidos de um grupo de mensagens SOAP arbitrário). A partir da mensagem SOAP inicial, a PDDP pode então gerar um conjunto de mensagens perturbadas. O usuário testador deve fornecer essa mensagem inicial arbitrária. O testador pode especificar as operações a serem realizadas, para a geração de mensagens de teste ou utilizar uma configuração padrão. A configuração tem por objetivo selecionar os tipos de perturbações que se deseja aplicar e parâmetros para limitar o número de mensagens geradas, que pode ser muito grande. Cada princípio de perturbação pode ser aplicado com base na substituição ou na adição de um elemento XML à mensagem. O sistema pode então gerar as mensagens perturbadas, que serão utilizadas para testar o *Web Service*.

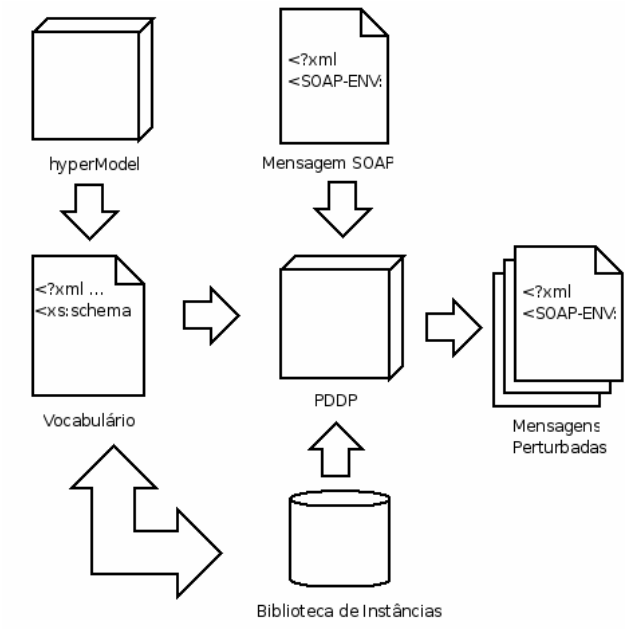



*Figura 8. 2: Diagrama de contexto da PDDP*

As mensagens perturbadas podem ser enviadas pela PDDP ao Web Service de interesse. É importante destacar que a avaliação da mensagem de resposta para cada mensagem perturbada é feita pelo testador (oráculo) e não pela ferramenta.

## 8.3 Arquitetura do Sistema

A PDDP foi desenvolvida sobre a tecnologia Java, que apresenta uma API[8] com um grande número de recursos para o tratamento de conteúdo XML [SUN06], incluindo RPC e SOAP. A Figura 8. 3 apresenta a arquitetura do sistema de forma simplificada, com a representação num diagrama de classes da UML.

Algumas observações devem ser feitas. 1) As classes representadas omitem detalhes irrelevantes da implementação; 2) as classes não apresentam um mapeamento direto para as unidades de implementação (uma classe do modelo UML pode ter sido mapeada para mais de uma classe Java, as quais não são representadas por se relacionarem a detalhes de implementação); 3) Os nomes das classes foram escritos com nomenclatura em português, por

[8] *Application Programming Interface* (Interface de Programação de Aplicativos)

conveniência de leitura (por exemplo, a classe Java PerturbadorAssociacoesHomologas é representada como Perturbador Associações Homólogas); 4) Foram omitidas as associações da classe Façade, para não poluir o diagrama (ver seção 8.3.9).

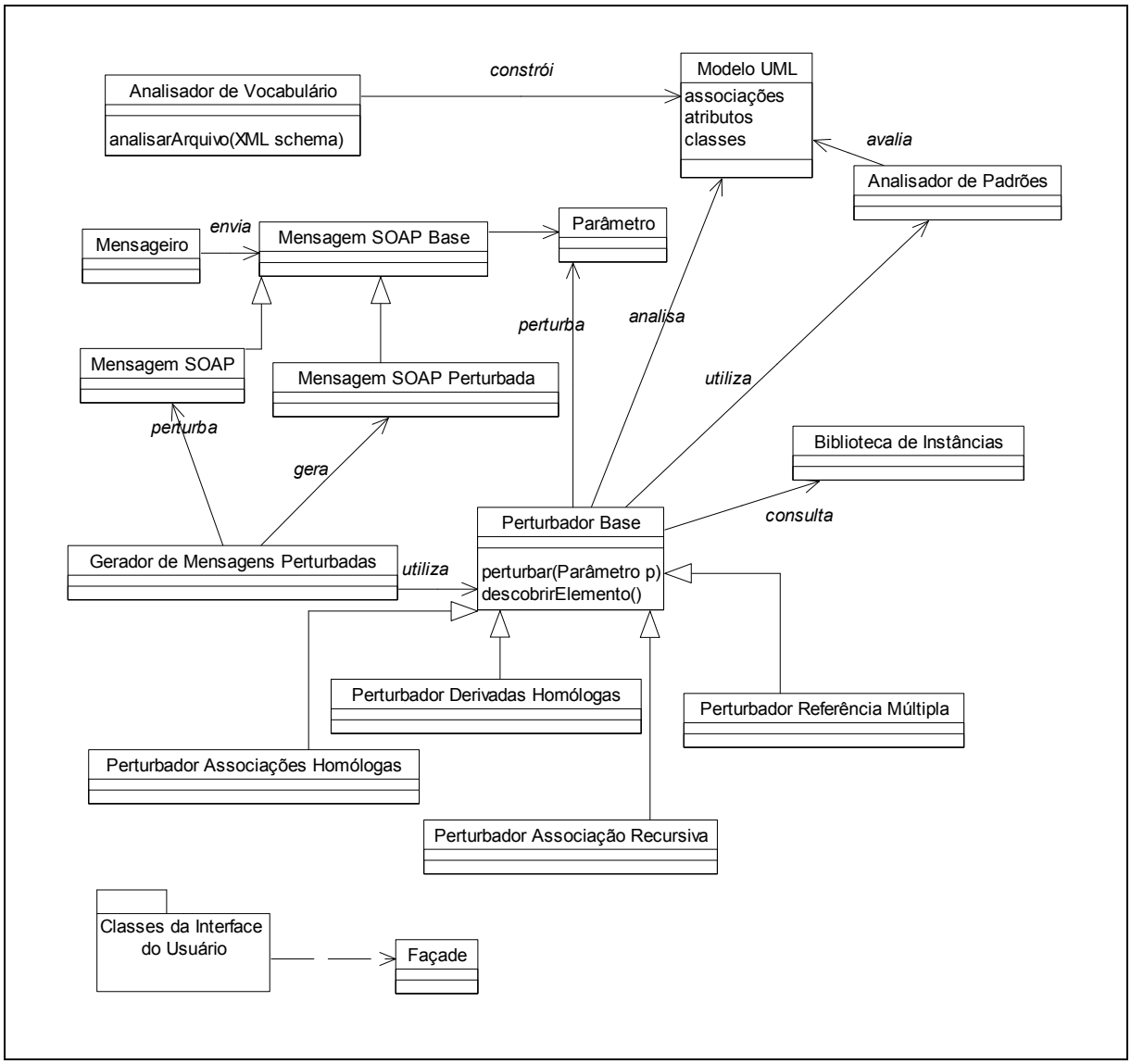



*Figura 8. 3: Arquitetura do Sistema*

A seguir são descritos os módulos componentes da arquitetura, mostrando como a interação entre os componentes permite a geração de dados de teste.

### 8.3.1 Analisador de Vocabulário

Como foi mostrado, o vocabulário da aplicação a ser testada é definido por meio de um modelo UML, que é construído com a ferramenta hyperModel. O modelo do diagrama de classes

pode ser transformado em um XML Schema, de acordo com um mapeamento específico [Car01]. O módulo Analisador de Vocabulário constrói uma representação do modelo UML, a partir da análise do XML Schema gerado pela ferramenta hyperModel.

### 8.3.2 Modelo UML

O Modelo UML criado com o Analisador de Vocabulário é formado por tabelas de classes, associações e atributos, que representam os elementos UML que compõem um modelo. O Modelo UML foi projetado de forma a facilitar a navegação entre os elementos, por meio de referências entre os objetos. A navegação por meio de referências permite uma boa análise do modelo, trabalho que seria dificultado se a árvore XML do esquema fosse analisada diretamente.

### 8.3.3 Analisador de Padrões

Essa classe avalia um modelo UML, verificando quais padrões foram encontrados para uma determinada classe UML. Não é avaliado o modelo como um todo, mas sim classes individuais, que são de interesse para a geração de mensagens perturbadas.

### 8.3.4 Mensagens SOAP e Parâmetro

Dois tipos de classe mensagem são derivados de uma classe base: Mensagem SOAP e Mensagem SOAP Perturbada. As classes de mensagem encapsulam o tratamento da string SOAP, apresentando métodos para a realização do *parse* do documento XML, assim como para o tratamento dos parâmetros da mensagem.

Uma classe Parâmetro possui informações sobre o elemento XML da mensagem SOAP que representa um parâmetro da RPC. Também possui informações específicas do modelo UML, como por exemplo a associação que referencia a classe que representa o parâmetro no modelo. Portanto a classe Parâmetro representa o parâmetro do *Web Service* mais meta-informação do modelo UML.

Por sua vez, a classe Mensagem SOAP Perturbada apresenta meta-informação sobre como o parâmetro foi perturbado (gerado). Também apresenta métodos para modificar a mensagem, como por exemplo a substituição do valor de um parâmetro.

### 8.3.5 Biblioteca de Instâncias

A Biblioteca de Instâncias abstrai um conjunto de documentos XML que possui instâncias de elementos definidos segundo o vocabulário. Os valores dos elementos XML da biblioteca, que devem ser consistentes com o XML Schema, são utilizados para a geração de dados de teste. A escolha dos elementos envolve a semântica das estruturas das árvores XML disponíveis. O usuário é responsável por prover uma biblioteca de instâncias, sendo que a qualidade dos dados de teste gerados depende da disponibilidade de uma boa biblioteca.

### 8.3.6 Perturbadores

Foram implementados quatro tipos de perturbadores, que estendem uma classe base. Um perturbador aplica o algoritmo de perturbação a um parâmetro da mensagem, sendo que a classe Gerador de Mensagens Perturbadas é responsável por utilizar as perturbações em conjunto, no contexto da mensagem SOAP como um todo. Foram definidos algoritmos de perturbação que refletem os princípios propostos na perturbação de dados dirigida por padrões. Dessa forma, os perturbadores utilizam o módulo Analisador de Padrões para avaliar como será o tratamento de um parâmetro, assim como consultam a Biblioteca de Instâncias para escolher os melhores dados de teste disponíveis.

Cada perturbador apresenta dois métodos principais: o método para perturbar um parâmetro e o método para descobrir um elemento de interesse (não utilizado somente no Perturbador Associações Recursivas). Durante a perturbação de um parâmetro é gerada uma lista de valores alternativos para esse parâmetro. Para perturbar um parâmetro é necessário definir um elemento de interesse, por exemplo: qual das associações homólogas se relaciona ao parâmetro; ou qual das associações de uma estrutura de referência múltipla está referenciando o parâmetro. Para descobrir o elemento de interesse são utilizadas heurísticas, pois na maioria das vezes as informações presentes na mensagem SOAP se apresentam ambíguas, no contexto do vocabulário da aplicação (o XML Schema). As heurísticas pesquisam o vocabulário de instâncias para descobrir um valor inicial para o elemento de interesse (sua classe no diagrama de classes), que será apresentado ao usuário se ele desejar confirmar a precisão da heurística, tendo a opção de modificar o valor sugerido. As heurísticas são bastante simples e se restringem a comparar o valor dos parâmetros da mensagem SOAP ao valor dos elementos encontrados na biblioteca. Caso sejam encontrados valores semelhantes, assume-se que os relacionamentos que podem ser descobertos a partir da biblioteca são aplicáveis à mensagem SOAP. Como foi comentado, o

sistema permite ao usuário interferir na precisão dessa escolha, caso o usuário deseje informar o relacionamento exato entre os elementos.

### 8.3.7 Gerador de Mensagens Perturbadas

Esse módulo é responsável por processar uma mensagem SOAP de entrada, das quais irá derivar mensagens perturbadas. Para isso são utilizadas as classes perturbadoras dos parâmetros da mensagem original, que geram valores alternativos para o parâmetro. Com um valor alternativo podem ser realizadas duas operações sobre a mensagem perturbada: a substituição do valor do parâmetro original ou a adição de um parâmetro perturbado à mensagem. Além disso pode-se aplicar a perturbação de diversas maneiras, por exemplo: perturbar um parâmetro em cada mensagem; perturbar mais de um parâmetro de uma mensagem; ou gerar combinações entre as listas de valores alternativos.

Essas diversas opções que definem a geração de mensagens perturbadas definem o número de mensagens geradas. Por exemplo, se todas as combinações de valores de parâmetros da mensagem SOAP perturbados forem aplicadas a uma mensagem, o número de mensagens perturbadas geradas pode ser intratável. Por isso a ferramenta permite que o usuário configure o modo de geração das mensagens e os padrões aplicados, assim como apresenta controles para restringir o número máximo de mensagens geradas (por exemplo definindo a profundidade máxima de combinação dos valores alternativos ou a seleção aleatória de um número máximo de valores gerados). A configuração padrão aplica todos os padrões, perturbando apenas um parâmetro de cada mensagem. Assim não são geradas combinações de parâmetros perturbados por padrão, por exemplo, gerar todas as mensagens com um parâmetro pertubado, todas as mensagens com dois parâmetros perturbados e assim por diante. Quanto maior esse nível (o número de parâmetros perturbados), maior o número de combinações, pois assim se multiplica o conjunto de padrões aplicáveis a cada parâmetro pelo número de parâmetros em cada nível. A geração de um grande número de mensagens por meio dessa combinação pode ser útil se a biblioteca de instâncias for muito pequena, oferecendo poucos valores para a substituição dos parâmetros.

### 8.3.8 Mensageiro

Depois de geradas as mensagens perturbadas, o testador poderá enviá-las para um *Web Service*, a fim de avaliar os resultados. O módulo Mensageiro é responsável por tratar da comunicação com o *Web Service*, possibilitando a configuração do serviço, o envio e recebimento

de mensagens SOAP. A classe foi baseada no *framework* Axis - Apache EXtensible Interaction System [Apa05.c].

### 8.3.9 Façade e Classes da Interface do Usuário

Para que o testador possa utilizar o sistema, foram desenvolvidas diversas classes que implementam a interface do usuário. Para prover uma visão única que abstrai os diversos subsistemas apresentados anteriormente, foi utilizado o padrão Façade [GHJV95], que define métodos de alto nível para as classes da camada de apresentação. A Figura 8. 4 mostra a tela principal da ferramenta PDDP. No Apêndice C são mostradas as telas com os passos fundamentais para a execução dos testes.

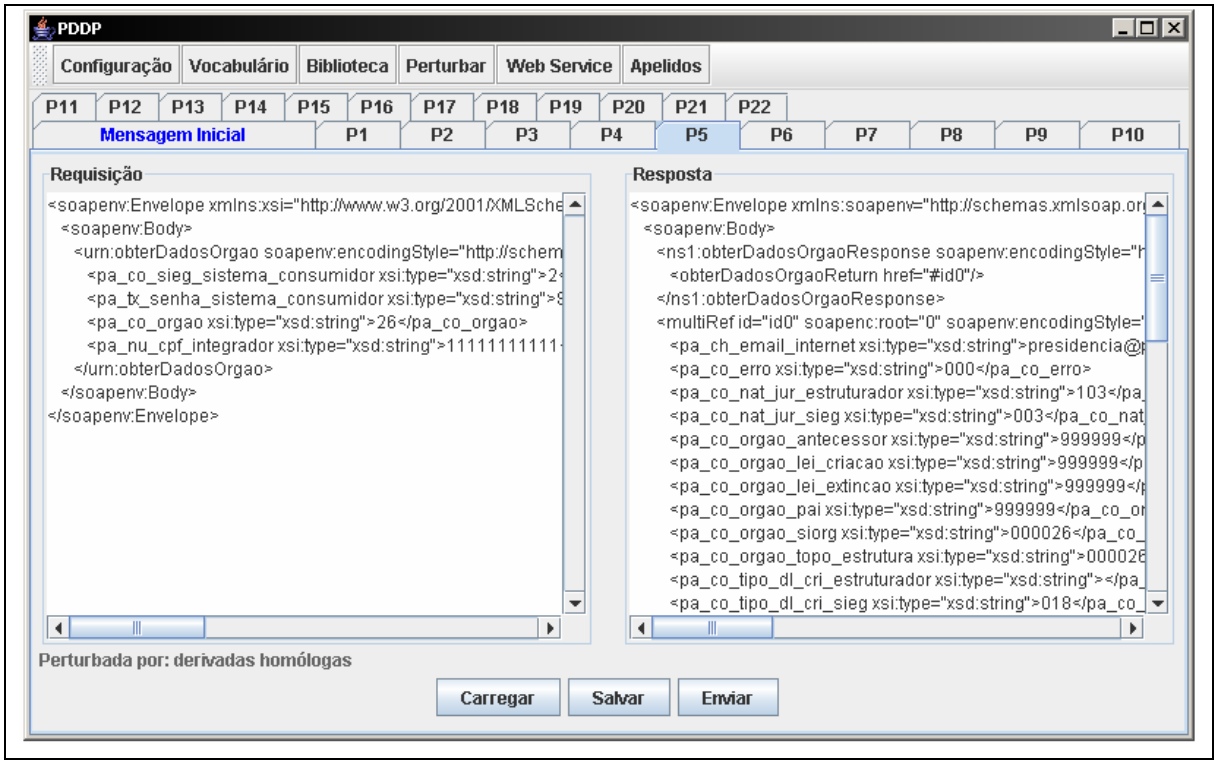



*Figura 8. 4: Tela principal da ferramenta*

## 8.4 Interação entre os componentes

Para mostrar como os componentes da arquitetura interagem entre si, partindo de eventos do usuário, será mostrado um modelo simplificado de como funciona a ferramenta. O diagrama de

colaboração da Figura 8. 5 abstrai a troca de mensagens entre os componentes (de fato, o número de mensagens trocadas entre os objetos na implementação é muito maior).

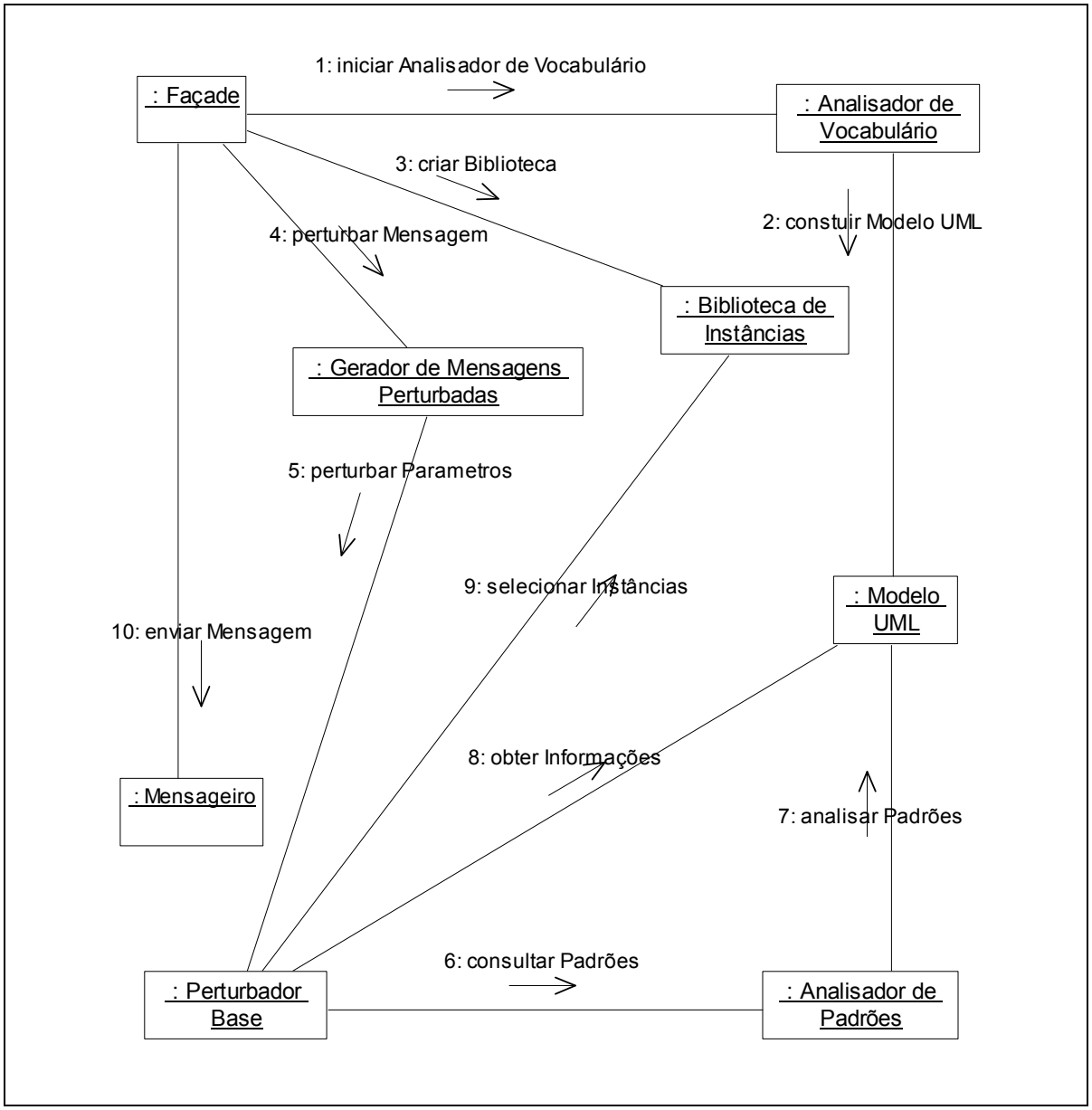



*Figura 8. 5: Diagrama de colaboração da ferramenta*

O objeto Façade representa a origem dos eventos do testador, que são disparados da interface do usuário. As mensagens estão numeradas, de acordo com a ordem em que devem ocorrer numa interação completa com a ferramenta. O resultado dessa interação é a geração de mensagens perturbadas que poderão ser enviadas ao *Web Service* de interesse, a fim de testá-lo.

A primeira mensagem (iniciar Analisador de Vocabulário) mostra que o usuário deve informar à ferramenta qual o vocabulário da aplicação. Para isso o objeto Analisador de Vocabulário irá processar um arquivo XML que contém o XML Schema construído com a ferramenta hyperModel. O resultado dessa operação é a criação de um modelo UML, conforme a segunda mensagem (criar Modelo UML).

A terceira mensagem (criar Biblioteca) representa a operação que o usuário deve realizar para a criação da biblioteca de instâncias, onde são fornecidos um ou mais arquivos XML ao objeto Biblioteca de Instâncias.

Ao final dessas operações será possível perturbar a mensagem, ação representada na quarta mensagem (perturbar Mensagem). Nesse ponto se inicia uma colaboração entre os principais objetos da ferramenta. O objeto Gerador de Mensagens Perturbadas, de acordo com a quinta mensagem (perturbar Parâmetros), irá solicitar que o objeto Perturbador Base[9] realize operações de perturbação nos parâmetros da mensagem que necessitam ser perturbados.

Para perturbar os parâmetros, o objeto Perturbador Base começa consultando informações sobre os padrões encontrados no modelo, por meio da sexta mensagem (consultar Padrões) enviada ao objeto Analisador de Padrões. O analisador deverá consultar o modelo UML construído inicialmente, conforme mostrado na sétima mensagem (obter Informações). De forma semelhante, o objeto Perturbador Base também deverá consultar informações sobre o modelo UML (oitava mensagem).

Nesse momento o perturbador tem as informações necessárias para selecionar instâncias da Biblioteca de Instâncias, as quais serão utilizadas para perturbar os parâmetros; a nona mensagem (selecionar Instâncias) mostra essa operação. As instâncias são selecionadas conforme os padrões a serem aplicados na perturbação (que podem ser escolhidos pelo usuário) e também com base em restrições, como um numero máximo de perturbações ou a seleção aleatória de elementos num grande conjunto (o que pode ser configurado pelo testador).

[9] O objeto Perturbador Base aqui representa cada uma das implementações específicas dos perturbadores.

Com os valores perturbados de cada parâmetro disponíveis, o objeto Gerador de Mensagens Perturbadas poderá gerar as mensagens, de acordo com a configuração informada pelo usuário e as heurísticas utilizadas pelo perturbador. Ao fim desse processo, o testador poderá enviar as mensagens perturbadas ao *Web Service* para analisar os resultados, ação representada na décima mensagem (enviar Mensagem), na qual o objeto Mensageiro é responsável pela comunicação. O testador deve então avaliar a resposta, verificando se o *Web Service* respondeu corretamente. A ferramenta não se preocupa em avaliar a resposta, pois a figura do oráculo é representada tão somente pelo testador. O problema de criar oráculos automatizados está fora do escopo da ferramenta, que se concentra mais na geração dos dados de teste.

## 8.5 Considerações Finais

Esse capítulo apresentou os aspectos técnicos relativos à implementação da ferramenta proposta a ser utilizada na validação dos aspectos teóricos da perturbação de dados dirigida por padrões. A PDDP possibilitou uma avaliação dos conceitos propostos, pois permitiu uma aplicação prática do uso dos padrões definidos. Assim pôde-se verificar que os conceitos são implementáveis e aplicáveis em um ambiente real.

A ferramenta foi utilizada em um estudo de caso, que será mostrado no próximo capítulo, quando serão discutidas a eficiência e as limitações encontradas. O estudo de caso vem a validar a aplicação da ferramenta num ambiente menos restrito que o ambiente de implementação (que pode ser tornar muito dirigido aos aspectos teóricos), pois esse estudo foi feito com a aplicação da ferramenta num ambiente de uso real de *Web Services*.

*Opostos não são contraditórios, mas complementares.*
***N. Bohr***

# *9 Estudo de Caso*

A ferramenta apresentada no capítulo anterior mostrou como os aspectos teóricos da perturbação de dados dirigida por padrões são implementáveis, assim como revelou as dificuldades e elementos necessários à implementação. Entretanto é necessário validar a sua utilização em um ambiente real.

Nesse capítulo são mostrados os detalhes de um experimento realizado com a ferramenta, que foi utilizada para o teste de um sistema composto por nove *Web Services* que são utilizados na integração de sistemas do governo federal, desenvolvidos em diversas plataformas diferentes.

A seguir são mostrados os detalhes do ambiente em que foi utilizada a PDDP, ainda no momento inicial do desenvolvimento do sistema integrador. Em seguida é apresentada a metodologia empregada na execução do experimento, assim como as dificuldades encontradas. Ao final são analisados os resultados obtidos, que foram comparados a resultados obtidos com outras abordagens.

## 9.1 *Web Services* Testados

O sistema testado é um sistema integrador de outros sistemas (chamados sistemas estruturadores), que são utilizados pelo governo federal para a administração de diversos órgãos[10]. No conjunto de sistemas estruturadores existem diversos tipos, desde sistemas legados, até sistemas que se encontram em desenvolvimento evolutivo. As tecnologias utilizadas na implementação desses sistemas são diversas, refletindo opções feitas ao longo de anos; temos assim sistemas executando em mainframes antigos, sistemas executando sobre arquitetura x86 desenvolvidos com software proprietário e, mais recentemente, sistemas desenvolvidos com base em software livre, o que reflete a recente política do governo federal.

[10] Devido a um acordo de não divulgação de informações, os detalhes de implementação dos sistemas utilizados nesse estudo de caso não foram mostrados. Entretanto essa omissão não prejudica o entendimento dos sistemas e não atinge informações relevantes para o experimento.

Ao longo do tempo esses diversos sistemas estabeleceram diversas formas para a troca de dados, de maneira desorganizada. O resultado foi um alto acoplamento entre as aplicações, o que tornou-se indesejável. O sistema integrador testado trata-se de um conjunto de *Web Services* que têm por objetivo integrar a comunicação entre os diversos sistemas estruturadores, de maneira centralizada. Para isso foi definida uma arquitetura baseada em eventos e troca de mensagens. A Figura 9. 1 apresenta um diagrama de pacotes que representa a integração entre os sistemas.

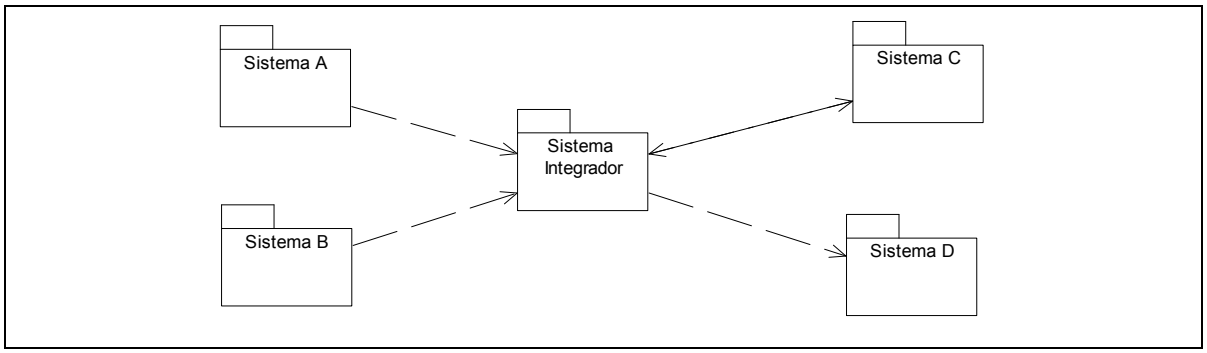



*Figura 9. 1: Diagrama de pacotes representando a integração entre os sistemas*

A versão do sistema integrador testada foi uma das versões iniciais, onde estão sendo integradas apenas informações sobre as estruturas de órgãos do governo federal. Futuramente pretende-se integrar informações sobre outras estruturas dos sistemas. Atualmente o sistema encontra-se em homologação, pois a sua implantação em ambiente de produção exigirá um grande esforço para a adaptação dos outros sistemas. Uma característica importante a ser destacada é que os erros encontrados durante o procedimento de testes são erros introduzidos durante o desenvolvimento, não existindo o caso de erros semeados.

A seguir serão descritos os *Web Services* do sistema integrador que foram testados.

- **obterDadosOrgao**: fornece informações sobre um determinado órgão do governo, as quais são controladas por um sistema estruturador. Devido a questões legais, a estrutura de órgão oficial deve ser mantida por um único sistema, para que os diversos sistemas que tratam de órgãos não venham a criar informações inconsistentes; assim o sistema integrador é a ponte entre os diversos sistemas e o sistema responsável pela informação oficial.
- **abrirListaEventosPendentes**: fornece uma lista de eventos para um determinado sistema estruturador. Sempre que houver mudanças na estrutura das informações dos sistemas é gerada uma lista de eventos, que determinam operações para que os sistemas se mantenham atualizados e sincronizados.

Note que um sistema estruturador é responsável por consultar se possui eventos pendentes, pois o sistema integrador não envia eventos para os sistemas estruturadores sem que eles tenham requisitado essa ação.

- **obterEventoDaListaPendencia**: Obtém os dados específicos sobre um evento contido na lista fornecida pelo *Web Service* anterior.
- **efetivarIntegracaoEventoSiorg**: esse *Web Service* tem a função de informar ao sistema integrador que um evento foi tratado, fornecendo informações sobre a integração do evento, as quais serão mantidas na base de dados do sistema integrador.
- **fecharListaEventosPendentes**: esse *Web Service* deve ser utilizado após a conclusão do processo de integração descrito por uma lista de eventos, o sistema estruturador deve informar ao sistema integrador de que a lista pode ser fechada.
- **obterDeParaOrgao**: por meio desse *Web Service* um sistema estruturador pode descobrir a correlação entre tabelas de diferentes sistemas que mantém a mesma informação em estruturas diversas. Por exemplo, um determinado órgão pode ter o código “x” no sistema A e código “y” no sistema B, apesar de se tratar do mesmo órgão (essas diferenças surgiram por motivos históricos, na evolução dos sistemas que possuem muitos anos). Portanto o sistema integrador responde a perguntas sobre informações da relação “de” uma tabela de um sistema “para” outra tabela de outro sistema.
- **atualizaDeParaOrgao**: a funcionalidade desse *Web Service* é semelhante ao caso anterior, com a diferença de que se tem a atualização de um mapeamento entre as tabelas dos sistemas, e não uma consulta. O mapeamento é mantido na base de dados do sistema integrador.
- **incluirEventoProduzido**: esse *Web Service* permite a um sistema estruturador gerar um evento, que é reflexo de uma mudança em alguma estrutura desse sistema. A partir dessa inclusão de evento, o sistema integrador irá decidir quais outros sistemas devem ser avisados sobre a mudança e gerar os eventos necessários para a atualização dos sistemas.

## 9.2 Metodologia

Para a aplicação dos conceitos de teste baseado em perturbação de dados dirigida por padrões são necessários os artefatos descritos no capítulo anterior: um vocabulário para o sistema, uma biblioteca de instâncias e mensagens XML a serem perturbadas. A seguir é discutida a metodologia adotada para a criação desses recursos, assim como as dificuldades encontradas.

### 9.2.1 Mensagens XML

Para cada um dos *Web Services* testados, foi fornecida à ferramenta uma mensagem SOAP inicial, de onde foram derivadas as mensagens perturbadas. A mensagem inicial é formada por um conjunto de parâmetros válidos, que podem ser enviados sem problemas aos *Web Services*.

### 9.2.2 Vocabulário da aplicação

O vocabulário da aplicação foi modelado utilizando-se a ferramenta hyperModel, conforme descrito no capítulo anterior. Entretanto nesse ponto surgiram algumas dificuldades a serem contornadas. Como descrito por Carlson [Car01], a maioria dos aplicativos não faz um bom uso dos recursos de modelagem de vocabulário que o XML Schema apresenta. Os aplicativos costumam utilizar os documentos XML apenas como uma lista de itens, sem semântica definida.

De fato, a pesquisa em torno de XML encontra-se à frente da tecnologia utilizada pelas empresas. Nesse caso observa-se que as técnicas propostas nessa pesquisa podem ser de difícil aplicação nos aplicativos comumente encontrados nas empresas, pois os conceitos empregados ainda são tema de pesquisa, notadamente a modelagem de vocabulários XML com o uso de UML. Para contornar esse problema, foi adotada a metodologia descrita a seguir.

Apesar de não estar explícito nos documentos XML trocados pelos *Web Services*, é claro o fato de que existe uma semântica bem definida entre os elementos. Entretanto essa semântica encontra-se no código fonte do aplicativo, conforme a sua documentação. Como a ferramenta para testes baseado em perturbação de dados necessita de um modelo UML que represente o vocabulário XML, a solução foi criar um modelo artificial, baseado na documentação do sistema integrador, ou seja nos documentos de diagramas de classe UML.

Para a criação desse modelo foram realizados os seguintes passos. Inicialmente foram levantados todos os parâmetros que fazem parte das mensagens dos *Web Services*. Em seguida foram analisados os diagramas de classes que documentam as entidades presentes no sistema, a fim de relacionar cada um dos parâmetros dos *Web Services* a uma das classes do sistema. Como resultado foi construído um diagrama de classes que representa o vocabulário do sistema para os parâmetros dos *Web Services*. Esse vocabulário foi modelado na hyperModel, para que pudesse

ser utilizado em conjunto com a PDDP. O modelo final, com simplificações, é apresentado na Figura 9. 2.

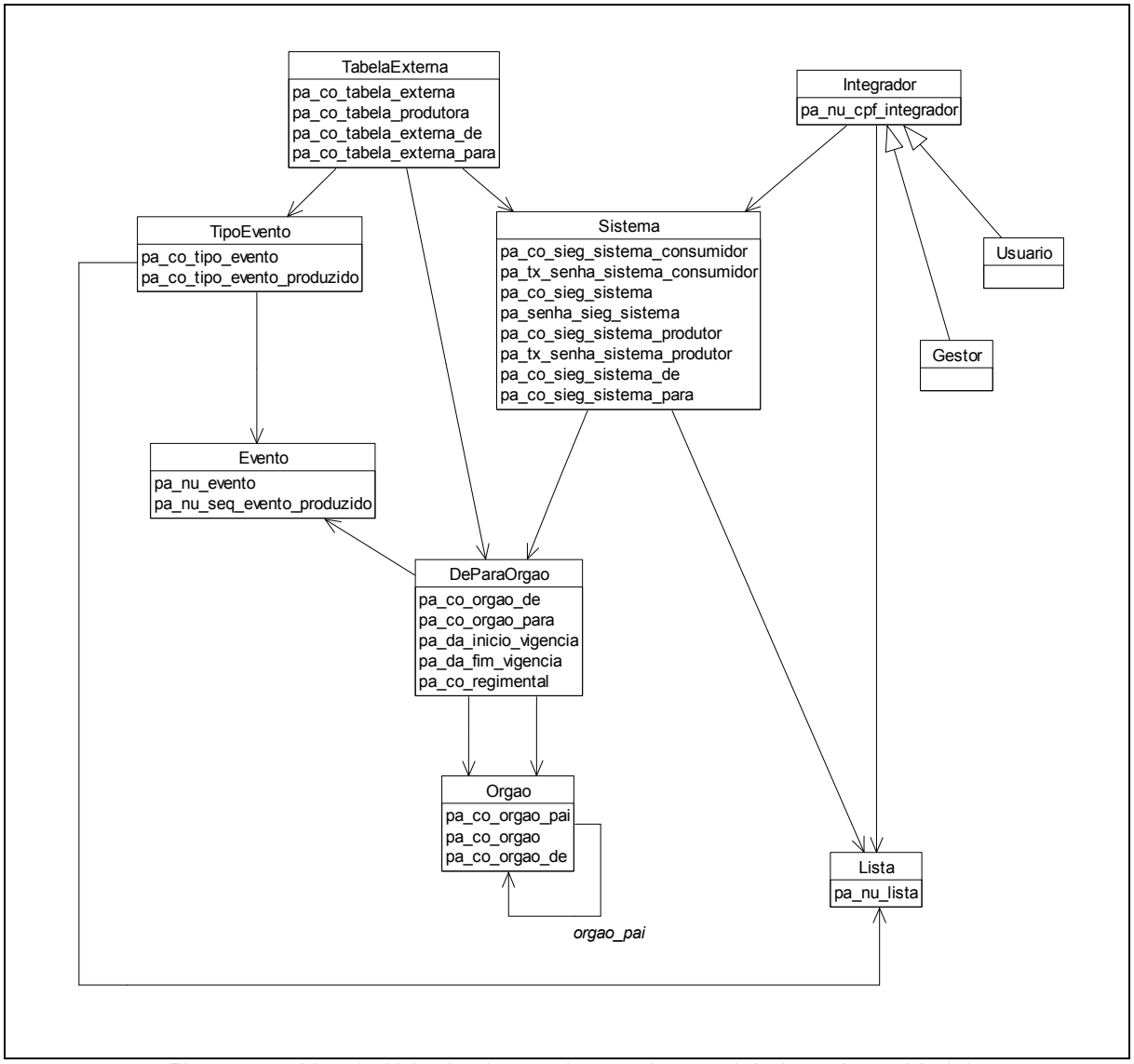



*Figura 9. 2: Vocabulário do sistema integrador, modelado na hyperModel*

Note que a semântica do vocabulário mantém-se fiel à semântica do sistema, pois o vocabulário foi obtido por meio da análise dos modelos UML que documentam o aplicativo. Por exemplo, veja como foi adicionada a associação recursiva da classe `Orgao`: observa-se que os *Web Services* possuem mensagens com os parâmetros `pa_co_orgao` e `pa_co_orgao_pai`; dessa maneira foi inserida a associação recursiva, que expressa a semântica de que um órgão possui um órgão-pai.

Outra observação a ser feita é que devido ao fato de a UML ser utilizada como linguagem para modelagem, tendo seus modelos definidos como resultado do processo de análise realizado por um humano, podem-se ter diferentes modelos para um mesmo sistema. Assim, uma característica importante da perturbação de dados dirigida por padrões é que as estruturas presentes nos modelos dependem do processo de análise do sistema, podendo ser definidas de diversas formas diferentes. Entretanto esse é um detalhe inerente ao processo de modelagem UML, do qual se depende.

Apesar da dificuldade em obter um vocabulário para o sistema, o vocabulário gerado é coerente com a semântica definida na documentação encontrada nos modelos UML do sistema integrador, sendo possível utilizá-lo de maneira satisfatória para o experimento. Vale a pena observar que a desvantagem de não se encontrar um modelo explícito para o caso dos *Web Services* é compensada pelo fato de que a metodologia de mapeamento UML/XML proposta por Carlson, assim como outras metodologias, são independentes da tecnologia que utiliza XML, nesse caso *Web Services*. Assim, a ferramenta de testes poderia ser aplicada, com pequenas modificações, para o teste de comunicação sob outros protocolos que se utilizam de XML.

### 9.2.3 Biblioteca de instâncias

Para criar a biblioteca de instâncias foi adotada a seguinte metodologia. A partir do vocabulário definido na hyperModel foram criadas estruturas de instâncias de elementos XML que obedecem à definição do vocabulário. Em seguida as estruturas foram populadas com valores obtidos de duas formas diferentes: 1) valores presentes no banco de dados do sistema integrador; 2) valores não presentes no banco de dados, mas que obedecem à estrutura do banco. As estruturas dos elementos XML foram preenchidas de maneira aleatória com esses valores.

Dessa maneira tem-se uma biblioteca de estruturas XML que reflete tanto os dados presentes no banco, quanto dados possíveis de serem armazenados no banco. Assim espera-se obter um vocabulário de boa qualidade, que represente a aplicação de forma efetiva.

Um aspecto a ser destacado é que o método de geração de mensagens perturbadas na perturbação dirigida por padrões considera valores coerentes com o uso real dos dados para os elementos XML, ao contrário dos métodos propostos em outras formas de perturbação, que tendem a gerar instâncias de documentos que dificilmente serão coerentes com a utilização proposta do aplicativo e como a estrutura do banco de dados [OL01, OW04]. Um exemplo é o operador `Divisão`, descrito no Capítulo 4, que troca o valor `v` de um elemento XML por `1/v`. Essa

abordagem tende a gerar dados de teste inconsistentes no âmbito da aplicação[11], o que não é o foco desse trabalho.

### 9.2.4 Realização dos testes

Com base nos artefatos descritos, foram realizados os testes com a utilização da ferramenta. Foram fornecidos dois arquivos XML de entrada: um XML Schema com a definição do vocabulário, gerado pela hyperModel; e um arquivo XML com instâncias de elementos válidos, segundo o XML Schema. Para cada *Web Service* foi fornecida uma mensagem inicial válida, da qual foram derivadas as mensagens perturbadas.

Para a geração das mensagens perturbadas, foi utilizada a configuração padrão da ferramenta, que perturba um parâmetro de cada vez e não gera combinação entre os parâmetros perturbados. A combinação de parâmetros gera um número muito elevado de mensagens, podendo ser utilizada em casos especiais. Os valores sugeridos pelas heurísticas que tentam descobrir dados sobre elementos ambíguos (cf. capítulo anterior) não foram modificados durante os testes. Portanto os resultados não foram influenciados pelo conhecimento do testador a respeito da aplicação, sendo utilizadas apenas as heurísticas presentes da ferramenta.

Os *Web Services* foram submetidos individualmente aos casos de teste gerados pela ferramenta. Durante a execução dos testes teve-se livre acesso ao banco de dados, para que pudessem ser verificados os resultados das operações. O resultado das operações depende tanto de pesquisas ao banco, para verificar o estado do sistema, quanto de escritas no banco, para a alteração do estado.

## 9.3 Resultados obtidos

Os resultados da execução dos casos de teste foram coletados relacionando-se as mensagens enviadas, os operadores de perturbação que deram origem às mensagens e os defeitos encontrados. A partir desse conjunto foram geradas as tabelas que seguem. A Tabela 9. 1 apresenta, para cada *Web Service*, o número de casos de teste gerados, o número de casos de

[11] Apesar disso, dados inconsistentes podem ser úteis para testes que buscam validar os parâmetros de entrada dos Web Services. Por exemplo, o operador `Divisão` pode gerar um valor fracionário a partir de uma chave-primária para o banco de dados, o que não faz sentido para uma consulta SQL, mas pode fazer sentido para a validação da chave-primária.

teste que revelaram defeitos e o total de defeitos encontrados. Note que o mesmo defeito foi revelado por mais de um caso de teste, em alguns *Web Services*.

*Tabela 9. 1: Casos de teste e defeitos*

| ***Web Service*** | **Casos de teste** | **Casos que revelaram defeitos** | **Total defeitos encontrados** |
|---|---|---|---|
| abrirListaEventosPendentes | 16 | 4 | 2 |
| atualizaDeParaOrgao | 19 | 2 | 1 |
| efetivarIntegracaoEventoSiorg | 9 | 1 | 1 |
| fecharListaEventosPendentes | 13 | 3 | 1 |
| incluirEventoProduzido | 9 | 0 | 0 |
| obterDadosEvento | 9 | 1 | 1 |
| obterDadosOrgao | 14 | 2 | 2 |
| obterDeParaOrgao | 11 | 2 | 2 |
| obterEventoDaListaPendencia | 12 | 3 | 2 |
| **Total** | **112** | **18** | **12** |

A Tabela 9. 2 mostra a contribuição que cada operador apresentou na detecção de defeitos. A contribuição é definida como a razão entre o número total de casos de teste que foram gerados pelo operador e o número de casos de teste que revelaram defeitos. Por exemplo, para o *Web Service abrirListaEventosPendentes* o número de casos de teste gerados em função do operador Associação Recursiva foi 2, o número total de casos de teste que revelaram defeitos foi 4; portanto a contribuição do operador foi de 50%. Note que para esse *Web Service* a soma das contribuições totaliza 125%, o que mostra que a mesma mensagem perturbada pode ter sido gerada por diferentes operadores.

*Tabela 9. 2: Contribuição por operador*

| ***Web Service*** | **AH** | **AR** | **DH** | **RM** |
|---|---|---|---|---|
| abrirListaEventosPendentes | 25% | 50% | 0% | 50% |
| atualizaDeParaOrgao | 0% | 0% | 0% | 100% |
| efetivarIntegracaoEventoSiorg | 0% | 0% | 0% | 100% |
| fecharListaEventosPendentes | 0% | 0% | 0% | 100% |
| incluirEventoProduzido | 0% | 0% | 0% | 0% |
| obterDadosEvento | 0% | 0% | 0% | 100% |
| obterDadosOrgao | 50% | 50% | 0% | 0% |
| obterDeParaOrgao | 0% | 0% | 0% | 100% |
| obterEventoDaListaPendencia | 0% | 0% | 0% | 100% |
| **Média** | **8,33%** | **11,11%** | **0%** | **72,22%** |

Abreviações utilizadas para os nomes dos operadores: AH=Associações Homólogas; AR=Associação Recursiva; DH=Derivadas Homólogas; RM=Referência Múltipla.

A contribuição média de cada operador mantém-se coerente com o vocabulário do sistema, conforme a Figura 9. 2. O operador Referência Múltipla apresentou a maior contribuição (72,22%) pois se trata do padrão com maior incidência no modelo. Os operadores Associações Homólogas (8,33%) e Associação Recursiva (11,11%) apresentaram uma contribuição

considerável, pois esses padrões aparecem uma única vez no modelo, na classe `Orgao`. Já o operador Derivadas Homólogas apresentou uma contribuição zero para todos os casos. Entretanto essa baixa contribuição pode ser explicada pela natureza dos *Web Services*: o parâmetro perturbado pelo padrão Derivadas Homólogas é o CPF de um usuário, que está relacionado a uma senha; quando se modificou o número do CPF (e se manteve a senha), os *Web Services* retornaram uma mensagem de validação avisando um erro na senha ou no usuário. Esse comportamento explica a contribuição mínima do operador, pois o mecanismo de validação do usuário é simples e não apresentou defeitos.

A Tabela 9. 3 apresenta a eficiência de cada operador de perturbação. A eficiência é definida como a razão entre o número de casos de teste que revelaram defeitos para um determinado operador e o número total de casos de teste gerados por esse mesmo operador. A abreviação N/A indica que o critério eficiência não é aplicável para a célula (caso em que o operador não gerou nenhuma mensagem).

*Tabela 9. 3: Eficiência dos operadores de perturbação*

| ***Web Service*** | **AH** | **AR** | **DH** | **RM** |
|---|---|---|---|---|
| abrirListaEventosPendentes | 33,33% | 50% | 0% | 28,57% |
| atualizaDeParaOrgao | N/A | N/A | 0% | 13,33% |
| efetivarIntegracaoEventoSiorg | N/A | N/A | 0% | 20,00% |
| fecharListaEventosPendentes | N/A | N/A | 0% | 33,33% |
| incluirEventoProduzido | N/A | N/A | 0% | 0% |
| obterDadosEvento | N/A | N/A | 0% | 20% |
| obterDadosOrgao | 25% | 33,33% | 0% | 0% |
| obterDeParaOrgao | N/A | 0% | N/A | 28,57% |
| obterEventoDaListaPendencia | N/A | N/A | 0% | 37,50% |
| **Média** | **29,16%** | **27,77%** | **0%** | **20,14%** |

A eficiência fornece um bom critério para a avaliação dos operadores, pois considerar apenas o número de casos de teste gerados ou o número de defeitos encontrados não fornece informações suficientes para uma boa análise. A eficiência média de cada operador apresentou valores bastante próximos, exceto para o operador Derivadas Homólogas. Uma eficiência média próxima mostra que cada um dos operadores possui características semânticas que contribuem de maneira significativa na revelação de defeitos, não havendo um operador notadamente muito melhor que os outros. Esse resultado pode variar conforme a natureza dos sistemas, como mostrou o baixo desempenho do operador Derivadas Homólogas. Na próxima seção será comparada a eficiência dos operadores com a eficiência obtida utilizando-se outros métodos.

### 9.3.1 Comparação dos resultados com operadores de perturbação tradicionais

Almeida [Alm06] testou os mesmos *Web Services* com uma ferramenta que foi desenvolvida com base nos operadores de perturbação de Offutt e Lee [OL01] e Offutt e Wuzhi [OW04] (com pequenas modificações). A seguir será feita uma comparação dos resultados obtidos.

No Capítulo 7 foi discutido como os operadores de perturbação tradicionais não têm por objetivo principal a análise semântica do vocabulário XML, assim como foram mostrados argumentos para a proposta dos operadores semânticos baseados em padrões. Dessa maneira, os operadores tradicionais serão chamados de operadores léxico-sintáticos, a fim de contrapô-los aos operadores semânticos, nas comparações a seguir. Foram implementados nove operadores léxico-sintáticos, que serão referenciados como *O1, O2, ..., O9*. Os nomes dos operadores foram omitidos pois para a comparação é relevante a classe dos operadores e não os detalhes de implementação.

A Tabela 9. 4 mostra, nas células acinzentadas, o número de defeitos encontrados pelos operadores léxico-sintáticos, para cada *Web Service* testado. As células claras, nas duas últimas colunas, foram colocadas na tabela pela conveniência na comparação; elas mostram os resultados obtidos com os operadores semânticos, mesmos resultados mostrados na Tabela 9. 1. Para os operadores léxico-sintáticos, o somatório dos defeitos não retrata o resultado exato, pois o mesmo defeito foi encontrado por diversos operadores, sendo registrado diversas vezes na tabela.

*Tabela 9. 4: Comparação do número de defeitos encontrados*

| ***Web Service*** | Léxico-Sintáticos | | | | | | | | | Semânticos | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **O1** | **O2** | **O3** | **O4** | **O5** | **O6** | **O7** | **O8** | **O9** | **Casos que revelaram defeitos** | **Total de defeitos encontra-dos** |
| abrirListaEventosPendentes | 1 | 1 | 1 | 0 | 0 | 0 | 2 | 0 | 0 | 4 | 2 |
| atualizaDeParaOrgao | 4 | 5 | 7 | 7 | 0 | 4 | 5 | 7 | 5 | 2 | 1 |
| efetivarIntegracaoEventoSiorg | 3 | 3 | 4 | 7 | 0 | 1 | 4 | 4 | 4 | 1 | 1 |
| fecharListaEventosPendentes | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 1 |
| incluirEventoProduzido | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 |
| obterDadosEvento | 2 | 2 | 2 | 4 | 1 | 1 | 2 | 2 | 2 | 1 | 1 |
| obterDadosOrgao | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 |
| obterDeParaOrgao | 6 | 5 | 6 | 6 | 0 | 0 | 6 | 6 | 5 | 2 | 2 |
| obterEventoDaListaPendencia | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 5 | 0 | 3 | 2 |

Na maioria dos casos o número de defeitos encontrados pelo conjunto de operadores semânticos foi menor que o número de defeitos encontrados pelo conjunto de operadores léxico-sintáticos. Duas exceções a esse comportamento ocorreram com os *Web Services fecharListaEventosPendentes* e *obterDadosOrgao*. Em ambos os *Web Services* os operadores léxico-sintáticos não encontraram nenhum defeito. Por outro lado, os operadores semânticos

encontraram um defeito em *fecharListaEventosPendentes*, onde três casos de teste revelaram o defeito; e dois defeitos em *obterDadosOrgao*, onde cada defeito foi revelado por um caso de teste.

Um fator que prejudica a comparação é que existe um número diferente de operadores implementados para cada conjunto; existem nove operadores para o conjunto léxico-sintático e quatro operadores para o conjunto semântico. Para uma melhor comparação dos resultados, a Tabela 9. 5 apresenta o valor máximo, o valor mínimo e a mediana do número de erros encontrados para cada *Web Service* pelos operadores léxico-sintáticos. A mediana mostra a tendência central do número de defeitos encontrados para um conjunto de operadores; a média não foi utilizada porque o mesmo defeito foi revelado diversas vezes pelos operadores léxico-sintáticos. Além disso a mediana apresenta um valor inteiro e a média apresenta valores fracionários para esse conjunto. Nas duas últimas colunas são destacados os *Web Services* onde o número total de defeitos encontrados pelos operadores semânticos foi maior que o número máximo e maior que a mediana dos defeitos encontrados pelos operadores léxico-sintáticos.

*Tabela 9. 5: Comparação do número de defeitos encontrados – máximo, mínimo e mediana*

| ***Web Service*** | Léxico-Sintáticos | | | Semânticos | | Comparação | |
|---|---|---|---|---|---|---|---|
| | **Máx.** | **Mín.** | **Mediana** | **Casos que revelaram defeitos** | **Total de defeitos encontrados** | **Total > Máximo** | **Total > Mediana** |
| abrirListaEventosPendentes | 2 | 0 | 0 | 4 | 2 | | X |
| atualizaDeParaOrgao | 7 | 0 | 5 | 2 | 1 | | |
| efetivarIntegracaoEventoSiorg | 7 | 0 | 4 | 1 | 1 | | |
| fecharListaEventosPendentes | 0 | 0 | 0 | 3 | 1 | X | X |
| incluirEventoProduzido | 3 | 0 | 0 | 0 | 0 | | |
| obterDadosEvento | 4 | 1 | 2 | 1 | 1 | | |
| obterDadosOrgao | 0 | 0 | 0 | 2 | 2 | X | X |
| obterDeParaOrgao | 6 | 0 | 6 | 2 | 2 | | |
| obterEventoDaListaPendencia | 5 | 0 | 0 | 3 | 2 | | X |

Como os defeitos encontrados pelos operadores léxico-sintáticos foram registrados mais de uma vez na tabela, o valor máximo é um bom indicativo do número total de defeitos encontrados. Observa-se que o número máximo relatado é maior que o total de defeitos encontrados pelos operadores semânticos na maioria dos casos; as únicas exceções são os dois *Web Services* já comentados. Nas linhas destacadas da coluna que mostra “Total > Mediana”, o conjunto de operadores semânticos apresentou um melhor resultado sobre metade dos operadores léxico-sintáticos, individualmente. Isso mostra que a variação do número de defeitos encontrados pelos operadores léxico-sintáticos é bastante grande, o que faz com que o número de defeitos encontrados pelos operadores semânticos se aproxime da tendência central.

A Tabela 9. 6 apresenta a eficiência média para cada um dos operadores. Os operadores semânticos apresentam uma menor variação da eficiência, a menos do operador Derivadas

Homólogas. Já os operadores léxico-sintáticos apresentam uma maior variação da eficiência; Almeida comenta esses resultados em detalhes, analisando quais operadores se apresentaram eficientes ou ineficientes [Alm06].

*Tabela 9. 6: Eficiência média por operador (em %)*

| Semânticos | | | | Léxico-Sintáticos | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **AH** | **AR** | **DH** | **RM** | **O1** | **O2** | **O3** | **O4** | **O5** | **O6** | **O7** | **O8** | **O9** |
| 29,16 | 27,77 | 0 | 20,14 | 14,81 | 29,63 | 38,89 | 6,61 | 4,65 | 15,56 | 35,19 | 16,67 | 29,63 |

Nessa comparação é importante notar que a eficiência geral dos operadores semânticos se apresentou bastante próxima da eficiência dos operadores léxico-sintáticos. A média das médias (grande média) da eficiência dos operadores semânticos é de 19,26%; a grande média dos operadores léxico-sintáticos é de 21,29%. A diferença é de 2,03%, o que mostra que a eficiência dos operadores semânticos é comparável à eficiência dos operadores léxico-sintáticos, apesar de estar um pouco abaixo.

## 9.4 Perspectivas para os operadores semânticos

Em resumo pode-se afirmar que os operadores semânticos revelaram, de forma geral, um número menor de defeitos que os operadores léxico-sintáticos, apresentando uma eficiência bastante próxima. A seguir são discutidos os aspectos desse resultado.

Devido à natureza léxico-sintática, os operadores de perturbação tradicionais tendem a revelar um grande número de defeitos de validação léxico-sintática de parâmetros. Por outro lado o número de defeitos revelados na lógica de negócios do aplicativo é menor. Analise-se por exemplo o operador Nulo[12] cujo comportamento é substituir o valor de um elemento XML por uma String vazia. Para o caso dos *Web Services* analisados, as instancias perturbadas pelo operador Nulo não poderão ser processadas pelo aplicativo num ciclo completo, pois os parâmetros são na maioria dos casos obrigatórios para a completa execução dos algoritmos. Nesse caso os defeitos mais prováveis de serem revelados são defeitos de validação, dado que a execução dos algoritmos não pode ser completa com um valor nulo. Um comportamento semelhante ocorre com os operadores Incompleto[13] e Espaço[14]. O operador Incompleto apaga uma sub-árvore do arquivo XML, o que deve ser validado pelo *parser* da árvore XML. O operador Espaço substitui o valor de um elemento por um caractere espaço.

[12] O3 nas tabelas
[13] O2 nas tabelas
[14] O7 nas tabelas

Por outro lado, os operadores semânticos têm o objetivo de gerar instâncias válidas, sem valores modificados em sua forma léxica ou sintática. Conforme foi mostrado anteriormente, de maneira especial no Capítulo 7, os operadores semânticos são dirigidos para a detecção de defeitos relativos à semântica do vocabulário da aplicação. Com a geração de instâncias válidas, têm-se muito mais chances de revelar erros da lógica de negócios do aplicativo. Nesse caso os defeitos encontrados na validação da formatação dos parâmetros são mínimos. Entretanto o número de defeitos encontrados pelos operadores semânticos provavelmente será menor em relação ao número de defeitos encontrados pelos operadores léxico-sintáticos, pois é muito mais difícil encontrar um defeito na lógica de negócios do aplicativo do que um defeito de validação de parâmetros.

Os resultados do experimento realizado apontam para essa característica dos operadores semânticos, que naturalmente não devem revelar tantos defeitos quanto os operadores léxico-sintáticos. Revelar erros na lógica de negócios dos aplicativos é algo mais ambicioso, representando um desafio maior. Respondendo a esse desafio, os operadores semânticos baseados em padrões demonstraram apresentar um boa contribuição, conforme as análises feitas na seção anterior.

Um exemplo que mostra a contribuição dos operadores semânticos pode ser observado na revelação de defeitos em dois *Web Services* onde os operadores léxico-sintáticos não conseguiram revelar nenhum defeito, conforme comentado anteriormente. Outro aspecto a ser considerado é a eficiência dos operadores semânticos, que se manteve próxima à eficiência obtida com operadores tradicionais; uma eficiência próxima indica que os operadores semânticos podem ser aplicados com um custo próximo aos operadores léxico-sintáticos.

Concluindo, pode-se afirmar que os dois conjuntos de operadores são complementares, onde os operadores semânticos vêm a contribuir para a revelação de uma classe de defeitos ainda não contemplada pelos operadores de perturbação tradicionais. Os defeitos revelados pelos dois conjuntos tendem a não constituir uma interseção, mas sim um complemento de defeitos mais voltados para a validação de parâmetros e de defeitos mais voltados para a validação da lógica de negócios do aplicativo, com atenção especial para o vocabulário.

## 9.5 Considerações Finais

Com a utilização da ferramenta em um ambiente real foi possível analisar como os conceitos teóricos podem ser aplicados. O estudo de caso permitiu a avaliação de diversos critérios utilizados para validar a utilização da ferramenta. As dificuldades inerentes à utilização da ferramenta puderam ser reveladas.

Esses aspectos serão úteis para a continuidade dessa pesquisa, pois a partir dos resultados encontrados e das dificuldades reveladas é possível propor novas abordagens baseadas nos conceitos teóricos desenvolvidos. As idéias para esses trabalhos futuros serão discutidas no próximo capitulo.

*Aprender a aprender é a habilidade mais importante na vida.*
***T. Buzan***

# *10 Conclusão e Trabalhos Futuros*

Essa dissertação explorou a concepção do uso de padrões de software aplicados a vocabulários XML, com aplicações para o teste de software. Foi introduzida uma abordagem baseada em perturbação de dados, utilizando conceitos de padrões XML e do mapeamento de domínio dos vocabulários XML para UML.

Os padrões utilizados foram definidos com ajuda de técnicas de mapeamento de documentos XML para diagramas UML, o que possibilitou explorar padrões de diferentes naturezas. A utilização de padrões aplicados a documentos XML e a aplicativos que se utilizam da XML foi pesquisada, a fim de propor abordagens que se aplicam ao contexto do teste de componentes que trocam mensagens XML.

Foi apresentado o uso de padrões de projeto de vocabulários XML, inspirados nos padrões da comunidade de software orientado a objetos. Foram mapeados padrões de projeto conhecidos ao contexto da modelagem de vocabulários, quando se encontraram restrições desse mapeamento, assim como vantagens dessa maneira de modelagem.

Padrões estruturais para vocabulários XML foram propostos, com o objetivo de realizar uma análise semântica dos vocabulários XML. Os padrões estruturais definidos constituem construções que comumente são encontradas em vocabulários que fazem uso da modelagem com UML. Para sua definição foram analisados vocabulários de aplicações XML reais.

Os padrões estruturais foram utilizados no auxílio à atividade de testes. Para isso foram estendidas técnicas de perturbação de dados, que buscam testar a comunicação de componentes que trocam mensagens XML. Assim foi possível propor a perturbação de dados dirigida por padrões. Essa abordagem permite a geração de dados de teste por meio de modificações nas mensagens XML. As modificações, que resultam da aplicação de operadores de mutação, costumam ser baseadas apenas em aspectos léxicos e sintáticos das mensagens. Nesse contexto, a aplicação de padrões estruturais XML a vocabulários XML, com ajuda do mapeamento UML/XML, possibilita uma análise semântica durante a geração de dados de teste.

Com base nessa proposta, foi implementada uma ferramenta, na qual os conceitos teóricos foram colocados em prática, a fim de possibilitar o teste de *Web Services*. A implementação da ferramenta mostrou que os conceitos de perturbação de dados dirigida por padrões podem ser aplicados a sistemas que se comunicam por meio de troca de mensagens XML arbitrárias, com base em um vocabulário XML. Para o caso específico de *Web Services*, foram levantadas as limitações e dificuldades para a implementação e aplicação dos aspectos teóricos.

A ferramenta foi aplicada a um sistema real, durante seu desenvolvimento e sem defeitos semeados, para validar a capacidade de revelação de defeitos da técnica. Com esse experimento pôde-se estudar a contribuição da técnica proposta, assim foi possível comparar seus resultados com os de outras técnicas existentes. Os resultados mostraram que a técnica é complementar às técnicas de perturbação de dados tradicionais, revelando defeitos que pertencem a uma classe que ainda não tinha sido explorada de maneira significativa por outros métodos.

São contribuições desse trabalho:

- Os padrões XML introduzidos, os quais facilitam:
    - A escrita dos documentos XML;
    - A compreensão dos aplicativos.
- A definição da perturbação de dados dirigida por padrões, que:
    - Permite considerar aspectos semânticos na geração de mensagens de teste;
    - Pode ser utilizada para o teste de componentes que trocam mensagens XML e que fazem uso de vocabulários;
    - Gera dados com probablidade de revelar outros tipos de defeitos complementares aos revelados pelo teste de perturbação tradicional.
- A ferramenta PDDP, que:
    - Torna a aplicação da abordagem prática para sistemas reais;
    - Valida as idéias introduzidas.
- O experimento mostra que:
    - Defeitos diferentes são revelados, comparados ao uso de operadores de perturbação tradicionais;
    - A abordagem é eficaz;
    - Defeitos relativos à semântica, mais difíceis de se encontrar, são revelados.

## 10.1 Trabalhos futuros

Um trabalho que pode ainda ser explorado é a definição de padrões de projeto XML por meio do uso de mapeamento UML. Esse estudo pode vir a contribuir com uma análise da possibilidade da aplicação de padrões de projeto XML, com base em padrões de projeto consolidados, como os padrões orientados a objetos. Nesse trabalho o conceito foi explorado de maneira superficial; na área de padrões, a contribuição da comunidade é bem-vinda, pois os padrões de projeto nascem de construções coletivas que se consolidam por meio da experiência dos projetistas.

De maneira semelhante, os padrões estruturais XML definidos nesse trabalho podem dar origem a outras construções baseadas nessa idéia. Nota-se que a maneira como os padrões estruturais são definidos é bastante arbitrária. Assim, pesquisas com outras abordagens podem vir a contribuir para o desenvolvimento de técnicas baseadas em padrões. Uma aplicação que pode ser citada é a análise da qualidade dos vocabulários XML, baseada em padrões.

Durante o desenvolvimento da ferramenta foi revelada uma dificuldade em se aplicar a técnica de teste baseado em padrões: a não disponibilidade de um vocabulário XML para a aplicação. Isso se deve ao fato de que as aplicações comerciais costumam não se beneficiar da definição de vocabulários que se utilizam do poder de expressão da XML, ou de técnicas de mapeamento UML. Essa dificuldade dá idéias para a pesquisa em trabalhos que explorem padrões obtidos a partir da análise de outros artefatos. Uma alternativa é a análise de padrões no modelo do bando de dados, por exemplo, com a definição de padrões nos modelos entidade-relacionamento. A partir dos princípios definidos nessa pesquisa, poderiam ser exploradas técnicas muito semelhantes, com foco em modelos entidade-relacionamento. Pode-se mencionar o uso de perturbação de dados em aplicações que utilizam uma comunicação qualquer, na qual as mensagens apresentem ligação com um banco de dados relacional.

Outra dificuldade encontrada para a implementação da técnica de perturbação de dados baseada em padrões é a obtenção de valores para popular as mensagens XML perturbadas. Uma melhoria ou adaptação que pode ser realizada é utilizar a técnica como auxiliar a outras técnicas de teste de *Web Service*, que não possuam a necessidade de obtenção de dados válidos para a perturbação. Um exemplo hipotético é uma ferramenta que testa *Web Services* com base na análise do valor limite. Um módulo dessa ferramenta poderia implementar a perturbação de dados dirigida por padrões e obter valores para a perturbação a partir da comunicação de mensagens utilizada pela ferramenta, na aplicação da técnica que não necessita de valores válidos (análise do valor limite).

A aplicação da perturbação de dados dirigida por padrões pode ser realizada em outros ambientes além de *Web Services*. Conforme citado acima, uma possibilidade é a exploração dos conceitos voltando-se para modelos entidade-relacionamento. Ainda, pode-se explorar essa idéia em ambientes que não necessariamente utilizam troca de mensagens sobre uma rede para se comunicar. Uma idéia a ser explorada é a comunicação de módulos de um sistema por meio de XML, com mensagens definidas segundo algum vocabulário. Por exemplo, arquivos de configuração XML são utilizados de maneira bastante abrangente em aplicativos escritos na plataforma Java. Uma aplicação de teste baseado em padrões para esses ambientes pode vir a ser uma boa contribuição para um melhor desenvolvimento de aplicativos baseados em XML.

Essa pesquisa apresentou a aplicação de padrões XML, com o auxílio de mapeamento para UML, com uma abordagem voltada para o teste de aplicativos XML. A pesquisa mostrou um aspecto bastante experimental, onde a implementação e a aplicação dos conceitos teóricos revelou benefícios e dificuldades. Com base nos princípios explorados, os trabalhos futuros mencionados e outras idéias que vierem a surgir poderão contribuir para a consolidação do uso de padrões XML baseados em modelos.

# *Referências*


[AIS77] ALEXANDER, C., ISHIKAWA, S., SILVERSTEIN, M., JACOBSON, M., FIKSDAHL-KING, I., ANGEL S. *A Pattern Language*. Oxford University Press, New York, 1977.

[Alm06] ALMEIDA JUNIOR, L. F. *Explorando Teste Baseado em Perturbação no Contexto de Web Services.* Dissertação de Mestrado. Universidade Federal do Paraná, Departamento de Informática, Mestrado em Engenharia de Software, 2006.

[Apa05.a] THE APACHE SOFTWARE FOUNDATION. Apache Struts. Disponível em: http://struts.apache.org/

[Apa05.b] THE APACHE SOFTWARE FOUNDATION. Apache XML Project. Disponível em: http://xml.apache.org/

[Apa05.c] THE APACHE SOFTWARE FOUNDATION. Apache Axis. Disponível em: http://ws.apache.org/axis/

[Arc00] Arciniegas, F. *Design Patterns in XML Applications*. XML.com. O'Reilly Media, Inc., Janeiro, 2000. Disponível em: http://www.xml.com/lpt/a/2000/01/19/feature/index.html

[AS06] ALMEIDA JUNIOR, L. F., VERGILIO, S. R. Exploring Perturbation Testing for Web Services. IEEE International Conference on Web Services (ICWS06), 717-126, Outubro 2006.

[BBFJS06] Bieberstein, N., Bose, S., Fiammante, M., Jones, K., Shah, R. *Service-Oriented Architecture Compass*. Pearson, Upper Saddle River, 2006

[BC87] BECK, K., CUNNINGHAM, W. Using Pattern Languages for Object-Oriented Programs. Technical Report nº CR-87-43. *OOPSLA-87 workshop on the Specification and Design for Object-Oriented Programming*. Setembro, 1987

[BCFK99] BOOCH, G., CHRISTERSON, M., FUCHS, M., KOISTINEN, J. *UML for XML Schema Mapping Specification*. Rational White Paper, Dezembro, 1999.

[Bei90] BEIZER, B. *Software Testing Techniques*, 2nd ed.. Van Nostrand Reinhold, Inc, New York NY, 1990.

[BGR02] BIRD, L., GOODCHILD, A., ROUTLEDGE, N. UML and XML Schema. In *13th Australian Database Conference (ADC2002)*, p. 157–166. ACS, 2002.

[BKK03] BERNAUER, M., KAPPEL, G., KRAMLER, G. *Representing XML Schema in UML - A UML Profile for XML Schema*. Technical Report, http://www.big.tuwien.ac.at/research/publications/2003/1303.pdf, 2003.

[BKK04] BERNAUER, M., KAPPEL, G., KRAMLER, G. Representing XML Schema in UML - A Comparison of Approaches. *Proceedings of the International Conference on Web Engineering (ICWE)*, Munich, Germany, Julho 2004.

[Blo02] BLOOMBERG, J. *Testing Web Services Today and Tomorrow*. The Rational Edge E-zine for the Rational Community – 2002. Disponível em: http://www.therationaledge.com

[BMRS96] BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., STAL, M. *Pattern-Oriented Software Architecture*, John Wiley and Sons, 1996

[BRJ99] BOOCH, G., RUMBAUGH, J., JACOBSON, I. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[Car01] CARLSON, D. *Modeling XML Applications with UML*. Addison-Wesley, 2001.

[DLS78] DEMILLO, R. A., LIPTON, R. J., SAYWARD, F. G. Hints on test data selection: Help for the practising programmer. *IEEE Computer Magazine 11*, 4 (Abril 1978), 34 – 41

[DO91] DEMILLO, R. A., OFFUTT, A. J. Constraint-based automatic test data generation. *IEEE Transactions on Software Engineering*, 17(9):900-910, Setembro 1991.

[FCS00] FREYTAG, J. C., CONRAD, R., SCHEFFNER, D. XML Conceptual Modeling Using UML. In *19th International Conference on Conceptual Modeling (ER),* Salt Lake City, Utah, USA, volume 1920 of Springer LNCS, p. 558–571, 2000.

[FW88] FRANKL, P., WEYUKER, E. An Applicable Family of Data Flow Testing Criteria. *IEEE Transactions on Software Engineering 14*, 10, (Outubro 1988), 1483 – 1498

[GG75] GOODENOUGH, J. B., GERHART, S. L. Toward a theory of test data selection. In *Proceedings of the International Conference on Reliable Software* (1975), pp. 493 – 510

[GHJV95] GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[HM04] HAROLD, E. R., MEANS, W. S., *XML in a Nutshell*, 2nd ed., O'Reilly, 2004

[IEEE90] IEEE. IEEE Standard Glossary of Software Engineering Terminology, 1990. IEEE Std. 610.12-1990

[ISO86] ISO 8879: Information processing - Text and office systems - Standard Generalized Markup Language (SGML). Outubro 1986. International Organization for Standardization.

[Lai05] LAINEVOOL, T. XML Patterns.com Home Page. Disponível em: http://www.XMLPatterns.com/

[Lar98] LARMAN, C.. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, Prentice Hall, 1998

[Mal91] MALDONADO, J. *Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software*. Tese de doutorado, DCA/FEE/UNICAMP, 1991

[MB89] MCCABE, T. J., BUTLER, C. W. Design complexity measurement and testing. *Communications of the ACM 32*, 12 (Dezembro 1989), 1415 – 1425

[Mye79] MYERS, G. J. *The Art of Software Testing*. John Wiley, 1979

[OL01] OFFUTT, J., LEE, S. C. Generating Test Cases for XML-based Web Component Interactions Using Mutation Analysis. *The Twelfth IEEE International Symposium on Software Reliability Engineering (ISSRE '01),* Hong Kong, Novembro, 2001.

[OMG05] OBJECT MANAGEMENT GROUP. *XMI Mapping Specification*, v2.1. Disponível em: http://www.omg.org/technology/documents/formal/xmi.htm

[OW04] OFFUTT, J., WUZHI, X. Generating Test Cases for Web Services Using Data Perturbation. *Workshop on Testing, Analysis and Verification of Web Services*, Boston Mass, Julho, 2004.

[OWL05] OFFUTT, J., WUZHI, X., LUO, J. Testing Web Services by XML Perturbation. *IEEE International Symposium on Software Reliability Engineering*. Chicago Illinois, Novembro 2005.

[Pap93] PAPADIMITRIOU, C. *Computational Complexity*, 1st ed., Addison Wesley, 1993

[Pon01] PONISIO, M. L., ROSSI, G. XML Patterns. *8th Conference on Pattern Languages of Programs*, Setembro 11-15, 2001, Monticello, Illinois, USA

[Pre01] PRESSMAN, R. S. *Software Engineering: A Practitioner's Approach*, 5th ed. McGraw-Hill, 2001

[Pro02.a] PROVOST, W. UML For W3C XML Schema Design. Disponível em: http://www.xml.com/lpt/a/2002/08/07/wxs_uml.html

[Pro02.b] PROVOST, W. *Structural Patterns in XML*. XML.com. O'Reilly Media, Inc., Setembro, 2002. Disponível em: http://www.xml.com/lpt/a/2002/09/04/strucpatterns.html

[RW85] RAPPS, S., WEYUKER, E. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering 11,* 04 (Abril 1985), 367 – 375

[SUN06] SUN MICROSYSTEMS. Java Technology – XML. Disponível em: http://java.sun.com/xml/

[Tas02] TASSEY, G. The economic impacts of inadequate infrastructure for software testing. Tech. Rep., National Institute of Standards and Technology, Maio 2002. RTI Project Number 7007.001

[W3C02] W3C. *Web Services Activity*, Janeiro 2002. Disponível em: http://www.w3.org/2002/ws/

[W3C03] W3C. Simple Object Access Protocol (SOAP). *W3C Recommendation*, Junho 2003. Disponível em: http://www.w3.org/TR/soap/

[W3C04.a] W3C. Datatypes in XML. *W3C Recommendation*, 2001. Disponível em: http://www.w3.org/TR/xmlschema-2/

[W3C04.b] W3C. Extensible Markup Language (XML) 1.0 (Third Edition). *W3C Recommendation*, Fevereiro 2004. Disponível em: http://www.w3.org/XML/

[W3C04.c] W3C. XML Schema Part 0: Primer, 2nd ed. *W3C Recommendation*, Outubro 2004. Disponível em: http://www.w3.org/TR/xmlschema-0/

[W3C05] W3C. Extensible Markup Language (XML). Disponível em: http://www.w3.org/XML/

# *APÊNDICE A – Exemplos de Esquemas XML*

Este apêndice contém os esquemas XML correspondentes aos diagramas UML que foram utilizados como exemplos no Capítulo 6. Caracteres especiais e acentos encontrados nas figuras dos modelos UML foram substituídos, devido a limitações do editor de XML (por exemplo “ç” e “á” foram substituídos por “c” e “a”). Os modelos UML dos exemplos são modelos conceituais, pois se pretendeu mostrar exemplos simples, sem definições complexas que poluíssem o diagrama; dessa forma, os modelos XML apresentam, quando necessário, elementos padrão, caso tenham sido omitidos nos diagramas (por exemplo o tipo de dado ou a cardinalidade).

## A.1 Exemplo do Padrão Árvore

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema                              xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<!-- ================================================ -->
<!-- Package: Main  -->
<!-- ================================================ -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Carrinho  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="Carrinho" type="Carrinho" />

   <xs:complexType name="Carrinho">
      <xs:sequence>
         <xs:element name="ID" type="xs:string">
         </xs:element>

         <xs:element name="URLDaCompra" type="xs:string">
         </xs:element>

         <xs:element name="itensDoCarrinho" type="ItensDoCarrinho" minOccurs="0">
         </xs:element>

         <xs:element      name="produtosSimilares"      type="ProdutosSimilares"
minOccurs="0">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Item  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="Item" type="Item" />

   <xs:complexType name="Item">
      <xs:sequence>
         <xs:element name="codigo" type="xs:string">
         </xs:element>

         <xs:element name="quantidade" type="xs:string">
         </xs:element>

         <xs:element name="preco" type="Preco" minOccurs="0">
         </xs:element>
```

```
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: ItensDoCarrinho  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="ItensDoCarrinho" type="ItensDoCarrinho" />

   <xs:complexType name="ItensDoCarrinho">
      <xs:sequence>
         <xs:element name="item" type="Item" minOccurs="1" maxOccurs="unbounded">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Preco  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="Preco" type="Preco" />

   <xs:complexType name="Preco">
      <xs:sequence>
         <xs:element name="moeda" type="xs:string">
         </xs:element>

         <xs:element name="precoFormatado" type="xs:string">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: ProdutoSimilar  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="ProdutoSimilar" type="ProdutoSimilar" />

   <xs:complexType name="ProdutoSimilar">
      <xs:sequence>
         <xs:element name="codigo" type="xs:string">
         </xs:element>

         <xs:element name="titulo" type="xs:string">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: ProdutosSimilares  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="ProdutosSimilares" type="ProdutosSimilares" />

   <xs:complexType name="ProdutosSimilares">
      <xs:sequence>
         <xs:element  name="produtoSimilar"  type="ProdutoSimilar"  minOccurs="1"
maxOccurs="unbounded">
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

## A.2 Exemplo do Padrão Associação Recursiva

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema                              xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<!-- ================================================= -->
<!-- Package: Main  -->
<!-- ================================================= -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: CategoriaDaLoja  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="CategoriaDaLoja" type="CategoriaDaLoja" />

   <xs:complexType name="CategoriaDaLoja">
      <xs:sequence>
         <xs:element name="ID" type="xs:int">
         </xs:element>

         <xs:element name="nome" type="xs:string">
         </xs:element>

         <xs:element name="ordem" type="xs:int">
         </xs:element>

         <xs:element name="categoriasFilhas" type="CategoriaDaLoja" minOccurs="0"
maxOccurs="unbounded">
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

## A.3 Exemplo do Padrão Associação Bidirecional

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema                              xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<!-- ================================================= -->
<!-- Package: Main  -->
<!-- ================================================= -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: LinhaDeDespacho  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="LinhaDeDespacho" type="LinhaDeDespacho" />

   <xs:complexType name="LinhaDeDespacho">
      <xs:sequence>
         <xs:element name="ID" type="xs:string">
         </xs:element>

         <xs:element name="codigoDeStatus" type="xs:string">
         </xs:element>

         <xs:element name="quantidade" type="xs:string">
         </xs:element>

         <xs:element name="nota" type="xs:string">
         </xs:element>

         <xs:element       ref="UnidadeTratadoraDeTransporte"       minOccurs="0"
maxOccurs="unbounded">
         </xs:element>
```

```
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: UnidadeTratadoraDeTransporte  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element                                    name="UnidadeTratadoraDeTransporte"
type="UnidadeTratadoraDeTransporte" />

   <xs:complexType name="UnidadeTratadoraDeTransporte">
      <xs:sequence>
         <xs:element name="ID" type="xs:string">
         </xs:element>

         <xs:element name="codigoDeTipoDeUnidade" type="xs:string">
         </xs:element>

         <xs:element ref="LinhaDeDespacho" minOccurs="0" maxOccurs="unbounded">
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

## A.4 Exemplo do Padrão Associação Derivada Base

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema                                xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<!-- ================================================ -->
<!-- Package: Main  -->
<!-- ================================================ -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: EquipeID  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="EquipeID" type="EquipeID" />

   <xs:complexType name="EquipeID">
      <xs:sequence>
         <xs:element name="ID" type="xs:string">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: EquipeIDAssinalada  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element        name="EquipeIDAssinalada"        type="EquipeIDAssinalada"
substitutionGroup="EquipeID" />

   <xs:complexType name="EquipeIDAssinalada">
      <xs:complexContent>
         <xs:extension base="EquipeID">
            <xs:sequence>
               <xs:element ref="EquipeID" minOccurs="0" maxOccurs="unbounded">
               </xs:element>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

</xs:schema>
```

## A.5 Exemplo do Padrão Associações Homólogas

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema                              xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<!-- ================================================ -->
<!-- Package: Main  -->
<!-- ================================================ -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Imagem  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="Imagem" type="Imagem" />

   <xs:complexType name="Imagem">
      <xs:sequence>
         <xs:element name="URL" type="xs:string">
         </xs:element>

         <xs:element name="altura" type="xs:float">
         </xs:element>

         <xs:element name="largura" type="xs:float">
         </xs:element>

         <xs:element name="estaVerificada" type="xs:string" minOccurs="0">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Oferta  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="Oferta" type="Oferta" />

   <xs:complexType name="Oferta">
      <xs:sequence>
         <xs:element     name="imagemGrande"     type="Imagem"     minOccurs="0"
maxOccurs="unbounded">
         </xs:element>

         <xs:element      name="imagemMedia"      type="Imagem"      minOccurs="0"
maxOccurs="unbounded">
         </xs:element>

         <xs:element     name="imagemPequena"     type="Imagem"     minOccurs="0"
maxOccurs="unbounded">
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

## A.6 Exemplo do Padrão Atalho

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema                              xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<!-- ================================================ -->
<!-- Package: Main  -->
<!-- ================================================ -->
```

```
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: EstatisticasDeTrafego  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="EstatisticasDeTrafego" type="EstatisticasDeTrafego" />

   <xs:complexType name="EstatisticasDeTrafego">
      <xs:sequence>
         <xs:element name="valor" type="xs:string">
         </xs:element>

         <xs:element name="delta" type="xs:string">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: EstatisticasDeUso  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="EstatisticasDeUso" type="EstatisticasDeUso" />

   <xs:complexType name="EstatisticasDeUso">
      <xs:sequence>
         <xs:element   name="rank"   type="EstatisticasDeTrafego"   minOccurs="0"
maxOccurs="unbounded">
         </xs:element>

         <xs:element   name="visualizacoesDePagina"   type="VisualizacoesDePagina"
minOccurs="0" maxOccurs="unbounded">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: VisualizacoesDePagina  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="VisualizacoesDePagina" type="VisualizacoesDePagina" />

   <xs:complexType name="VisualizacoesDePagina">
      <xs:sequence>
         <xs:element name="porUsuario" type="EstatisticasDeTrafego" minOccurs="0"
maxOccurs="unbounded">
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

## A.7 Exemplo do Padrão Ciclo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema                              xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<!-- ================================================ -->
<!-- Package: Main  -->
<!-- ================================================ -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Assercao  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="Assercao" type="Assercao" />

   <xs:complexType name="Assercao">
      <xs:sequence>
```

```
         <xs:element name="versaoMaior" type="xs:string">
         </xs:element>

         <xs:element name="versaoMenor" type="xs:string">
         </xs:element>

         <xs:element name="ID" type="xs:string">
         </xs:element>

         <xs:element name="emissor" type="xs:string">
         </xs:element>

         <xs:element ref="grupo" minOccurs="0" maxOccurs="unbounded">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Evidencia  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="Evidencia" type="Evidencia" />

   <xs:complexType name="Evidencia">
      <xs:sequence>
         <xs:element name="referencia" type="xs:string">
         </xs:element>

         <xs:element ref="Assercao" minOccurs="0" maxOccurs="unbounded">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: ExpressaoDeDecisaoDeAutorizacao  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element                                name="ExpressaoDeDecisaoDeAutorizacao"
type="ExpressaoDeDecisaoDeAutorizacao" />

   <xs:complexType name="ExpressaoDeDecisaoDeAutorizacao">
      <xs:sequence>
         <xs:element name="fonte" type="xs:string">
         </xs:element>

         <xs:element name="decisao" type="xs:string">
         </xs:element>

         <xs:element name="acao" type="xs:string">
         </xs:element>

         <xs:element ref="Evidencia" minOccurs="0" maxOccurs="unbounded">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: grupo  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="grupo" type="grupo" />

   <xs:complexType name="grupo">
      <xs:sequence>
         <xs:element     ref="ExpressaoDeDecisaoDeAutorizacao"     minOccurs="0"
maxOccurs="unbounded">
```

```
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Global Element Declarations  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
</xs:schema>
```

## A.8 Exemplo do Padrão Derivadas Homólogas

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema                              xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<!-- ================================================ -->
<!-- Package: Main  -->
<!-- ================================================ -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Atividade  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element                 name="Atividade"                 type="Atividade"
substitutionGroup="ElementosExtensiveis" />

   <xs:complexType name="Atividade">
      <xs:complexContent>
         <xs:extension base="ElementosExtensiveis">
            <xs:sequence>
               <xs:element name="nome" type="xs:string">
               </xs:element>

               <xs:element name="condicaoDeJuncao" type="xs:string">
               </xs:element>

               <xs:element name="suprimirFalhaDeJuncao" type="xs:boolean">
               </xs:element>

               <xs:element      name="fonte"      type="Fonte"      minOccurs="0"
maxOccurs="unbounded">
               </xs:element>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: ElementosExtensiveis  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="ElementosExtensiveis" type="ElementosExtensiveis" />

   <xs:complexType name="ElementosExtensiveis">
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Fonte  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="Fonte" type="Fonte" substitutionGroup="ElementosExtensiveis"
/>

   <xs:complexType name="Fonte">
      <xs:complexContent>
         <xs:extension base="ElementosExtensiveis">
```

```
            <xs:sequence>
               <xs:element name="nomeDoLink" type="xs:string">
               </xs:element>

               <xs:element name="condicaoDeTransicao" type="xs:string">
               </xs:element>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:schema>
```

## A.9 Exemplo do Padrão Herança Associação (1)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema                              xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<!-- ================================================ -->
<!-- Package: Main  -->
<!-- ================================================ -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Captura  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element                    name="Captura"                    type="Captura"
substitutionGroup="ConteinerDeAtividades" />

   <xs:complexType name="Captura">
      <xs:complexContent>
         <xs:extension base="ConteinerDeAtividades">
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: ConteinerDeAtividades  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="ConteinerDeAtividades" type="ConteinerDeAtividades" />

   <xs:complexType name="ConteinerDeAtividades">
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: TratadorDeFalhas  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="TratadorDeFalhas" type="TratadorDeFalhas" />

   <xs:complexType name="TratadorDeFalhas">
      <xs:sequence>
         <xs:element ref="Captura" minOccurs="0" maxOccurs="unbounded">
         </xs:element>

         <xs:element           ref="ConteinerDeAtividades"           minOccurs="0"
maxOccurs="unbounded">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

</xs:schema>
```

## A.10 Exemplo do Padrão Herança Associação (2)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema                              xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<!-- ================================================= -->
<!-- Package: Main  -->
<!-- ================================================= -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: Alexa  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="Alexa" type="Alexa" />

   <xs:complexType name="Alexa">
      <xs:sequence>
         <xs:element    name="informacoesDeContato"    type="InformacoesDeContato"
minOccurs="0">
         </xs:element>
      </xs:sequence>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: InformacoesDeContato  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element      name="InformacoesDeContato"      type="InformacoesDeContato"
substitutionGroup="ServicoURL" />

   <xs:complexType name="InformacoesDeContato">
      <xs:complexContent>
         <xs:extension base="ServicoURL">
            <xs:sequence>
               <xs:element name="nomeDoProprietario" type="xs:string">
               </xs:element>

               <xs:element name="email" type="xs:string">
               </xs:element>
            </xs:sequence>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: ServicoURL  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="ServicoURL" type="ServicoURL" />

   <xs:complexType name="ServicoURL">
      <xs:sequence>
         <xs:element name="dadoURL" type="xs:string">
         </xs:element>

         <xs:element name="URLNavegavel" type="xs:string">
         </xs:element>

         <xs:element        name="alexa"        type="Alexa"        minOccurs="0"
maxOccurs="unbounded">
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

## A.11 Exemplo do Padrão Herança Multinível

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema                              xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<!-- ================================================= -->
<!-- Package: Main  -->
<!-- ================================================= -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: inteiro  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element name="inteiro" type="inteiro" />

   <xs:complexType name="inteiro">
      <xs:simpleContent>
         <xs:extension base="xs:decimal">
            <xs:sequence>
               <xs:element name="digitosDeFacao" type="xs:string">
               </xs:element>
            </xs:sequence>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: inteiroNaoPositivo  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element        name="inteiroNaoPositivo"        type="inteiroNaoPositivo"
substitutionGroup="inteiro" />

   <xs:complexType name="inteiroNaoPositivo">
      <xs:simpleContent>
         <xs:extension base="inteiro">
            <xs:sequence>
               <xs:element name="maxInclusivo" type="xs:string">
               </xs:element>
            </xs:sequence>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- Class: inteiroNegativo  -->
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <xs:element           name="inteiroNegativo"           type="inteiroNegativo"
substitutionGroup="inteiroNaoPositivo" />

   <xs:complexType name="inteiroNegativo">
      <xs:simpleContent>
         <xs:extension base="inteiroNaoPositivo">
            <xs:sequence>
               <xs:element name="maxInclusivo" type="xs:string">
               </xs:element>
            </xs:sequence>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
</xs:schema>
```

# *APÊNDICE B – Algoritmos de Perturbação*

Este apêndice apresenta os algoritmos utilizados nas perturbações de dados propostas. Os algoritmos estão escritos em pseudo-código, em alto nível. São omitidos detalhes da implementação, assim como não é utilizada uma tipagem forte.

## B.1 Algoritmo para Perturbação Associações Homólogas

```
AH(atributo, associaçãoReferenciadora)
{
       classePerturbada = atributo.classe

       conjuntoAssociações = {};
       classeReferenciadora = associaçãoReferenciadora.origem

       para cada ( associação em classeReferenciadora.associaçõesSaindo )
       {
              se ( associação.destino == classePerturbada
                     e
                    associaçãoReferenciadora != associação )
              {
                     conjuntoAssociações.adicionar(associação)
              }
       }

       conjuntoNodos = Obtenha nodos arbitrários que possuem o mesmo nome do
       atributo perturbado

       valoresPerturbados = {}

       para cada( nodo em conjuntoNodos )
       {
              se ( conjuntoAssociacoes.possui (nodo.pai )
              {
                     valoresPerturbados.adicionar( nodo.pai )
              }
       }
       retorne valoresPerturbados

}
```

## B.2 Algoritmo para Perturbação Associação Recursiva

```
AR(atributo)
{
       valoresPerturbados = {}

       nodosPais = {}
       nodosPais.adicionar( atributo.nomeClasse )

       associaçõesChegando = atributo.classe.associaçõesChegando
       associaçõesSaindo = atributo.classe.associaçõesSaindo
```

```
        para cada ( associação em associaçãoSaindo )
        {
                se ( associaçõesChegando.contem( associação ) )
                {
                        nodosPais.adicionar( associação )
                }

        }

        conjuntoNodos = Obtenha nodos arbitrários com o mesmo nome do atributo

        valoresPerturbados = {}
        para cada ( nodo em conjuntoNodos )
        {
                se ( nodosPais.contem ( nodo.pai ) e
                        não nodo.ehIrmao( atributo.nodo ) e
                        nodo.valor != atributo.valor
                )
                {

                        valoresPerturbados.adicionar( nodo )
                }
        }

        retorne valoresPerturbados

}
```

## B.3 Algoritmo para Perturbação Derivadas Homólogas

```
DH( atributo, classePerturbada )
{
        conjuntoNodos = Obtenha nodos arbitrários com o mesmo nome do atributo

        valoresPerturbados = {}

        para cada ( nodo em conjuntoNodos )
        {
                se( não ( nodo.pai.classe == classePerturbada ) )
                {
                        conjuntoAssociaçõesChegando =
classePerturbada.associaçõesChegando

                        se( não (conjuntoAssociaçõesChegando.contem ( nodo.pai ) ) ou
                        classePerturbada == atributo.classe )
                        {
                                valoresPerturbados.adicionar(nodo)
                        }
                }
        }

        retorne valoresPerturbados
}
```

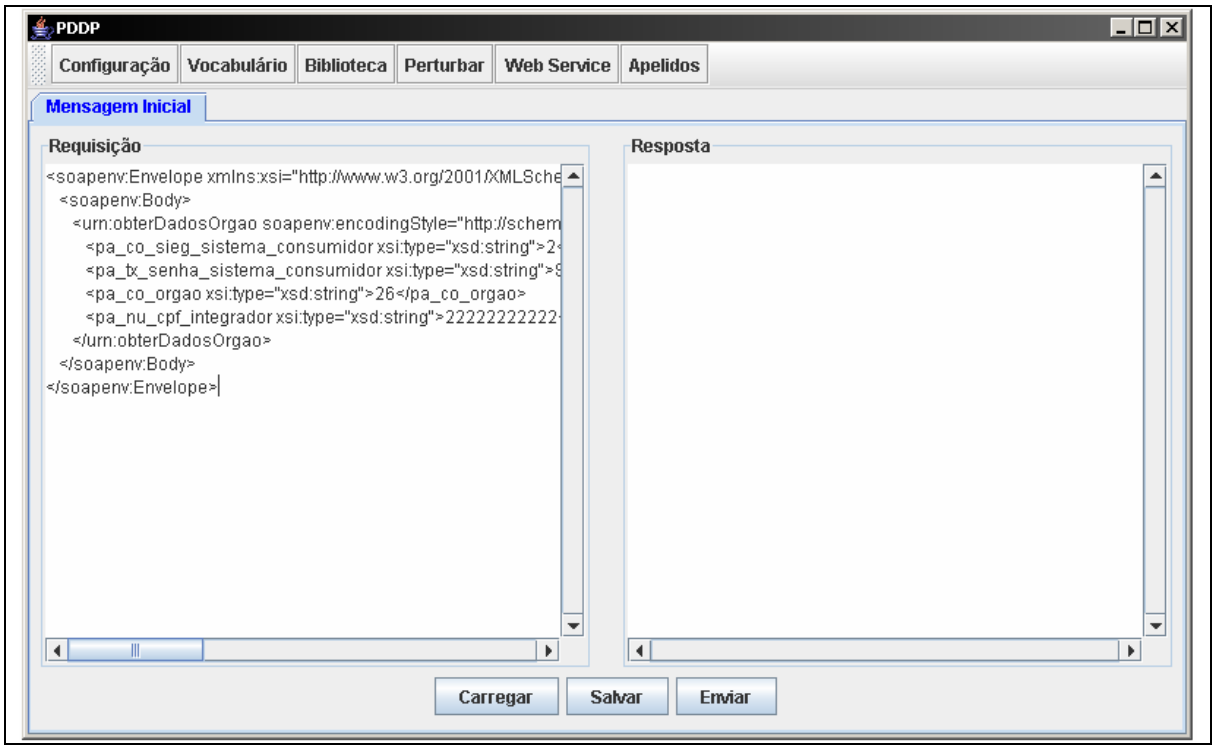
## B.4 Algoritmo para Perturbação Referência Múltipla

```
RM( atributo, classeReferenciadora )
{
       classePerturbada = atributo.classe

       conjuntoAssociações = {}
       para cada ( associação em classePerturbada.associaçõesSaindo )
       {
              se ( associação.origem != classeReferenciadora )
              {
                     conjuntoAssociações.adicionar( associação )
              }
       }

       conjuntoNodos = Obtenha nodos arbitrários com o mesmo nome do atributo

       valoresPerturbados = {}

       para cada ( nodo em conjuntoNodos )
       {
              se( conjuntoAssociações.contem ( nodo.pai ) )
              {
                     valoresPerturbados.adicionar( nodo )
              }
       }

       retorne valoresPerturbados

}
```

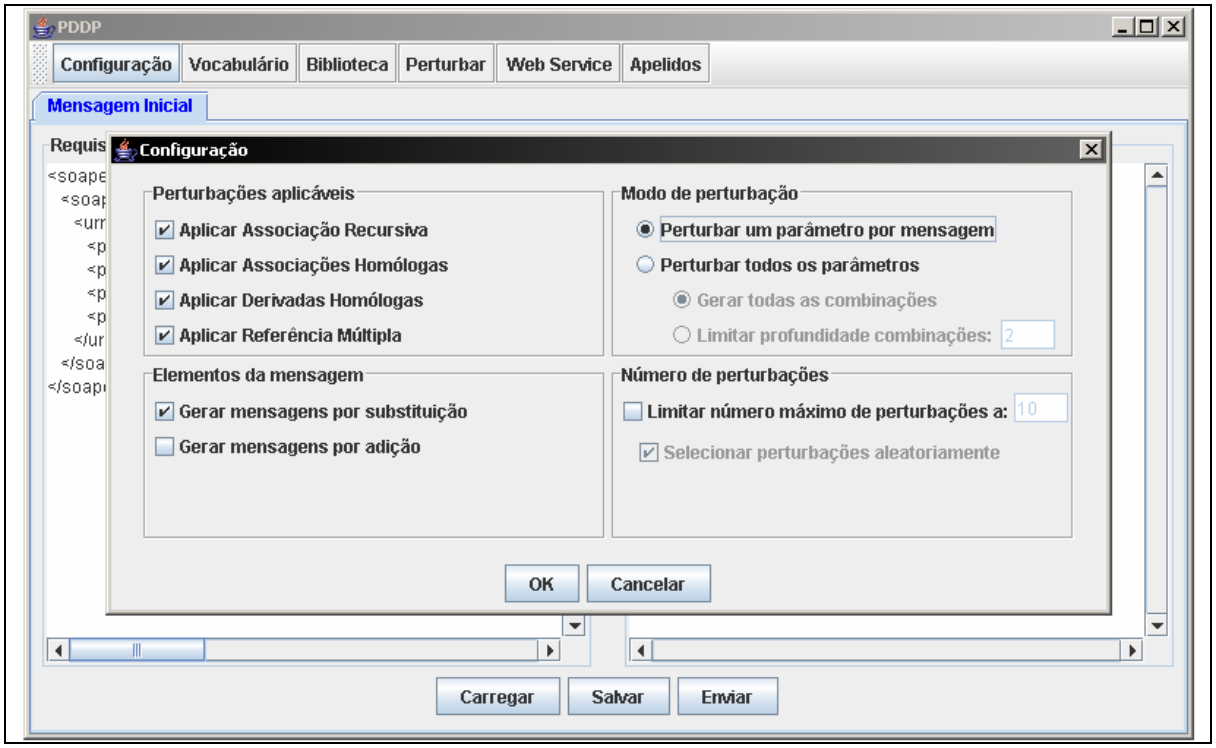
# *APÊNDICE C – Telas da Ferramenta PDDP*

Este apêndice apresenta telas da ferramenta PDDP, para mostrar os passos na execução dos testes (de forma resumida).

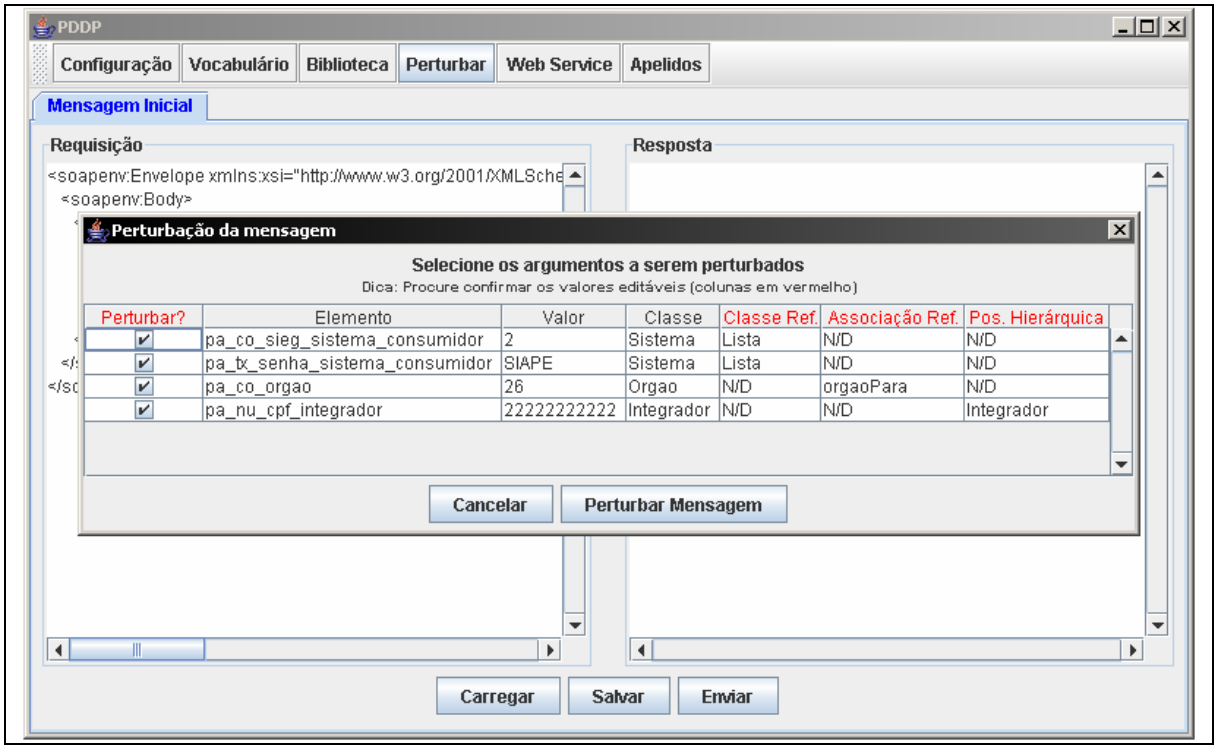
## C.1 Tela Inicial



## C.2 Diálogo de Configuração




## C.3 Diálogo de Perturbação da mensagem



## C.4 Comunicação de mensagem perturbada